精通Matlab与C/C++混合程序设计

（第2版）

刘 维 编著



北京航空航天大学出版社

CD-ROM INCLUDED

策划编辑：胡　敏
封面设计：runsign


ISBN 978-7-81124-178-5

定价：39.00元(含光盘)

精通 Matlab 与 C/C++ 混合程序设计

(第 2 版)

刘 维 编著

北京航空航天大学出版社

内 容 简 介

本书主要介绍如何运用 Matlab 与 C/C++进行混合程序设计。共分 8 章，主要包括：Matlab 程序设计初步、Matlab 编译器、Matlab 与 C 语言的接口、生成可独立运行的 Matlab 程序、Visual C++调用 Matlab 程序、Matlab DotNet Builder 与 Visual C ++、Matcom 与 C/C++以及 Visual C++调用 Matlab C++数学库。另外，附录中介绍有关动态链接库的基础知识。各章包含大量的实例程序，可供寻求将 Matlab 程序脱离 Matlab 环境的 Matlab 程序设计人员、寻求在 Matlab 中调用 C/C++程序的程序设计人员、寻求在 C/C++中调用 Matlab 程序的程序设计人员学习和参考。

本书附带一张光盘，其中包含各章实例程序的源代码。

图书在版编目(CIP)数据

精通 Matlab 与 C/C++混合程序设计/刘维编著. —2 版.
北京：北京航空航天大学出版社，2008.1
ISBN 978-7-81124-178-5

Ⅰ.精… Ⅱ.刘… Ⅲ.①算法语言—程序设计②C 语言—程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2007)第 172811 号

精通 Matlab 与 C/C++混合程序设计
(第 2 版)
刘 维 编著
责任编辑 胡 敏
*
北京航空航天大学出版社出版发行
北京市海淀区学院路 37 号(100083) 发行部电话：(010)82317024 传真：(010)82328026
http://www.buaapress.com.cn E-mail：bhpress@263.net
涿州市新华印刷有限公司印装 各地书店经销
*
开本：787×1 092 1/16 印张：22.75 字数：582 千字
2008 年 1 月第 1 版 2008 年 1 月第 1 次印刷 印数：5 000 册
ISBN 978-7-81124-178-5 定价：39.00 元(含光盘)

前 言

自 MathWorks 推出 Matlab 7.0 以后，Matlab 编译器在很多方面都发生了很大变化。其中最大的变化是 Matlab 编译器不再将 Matlab 程序直接编译为 C/C++代码，而只生成 Matlab 程序接口文件，Matlab 程序则直接交给 MCR(Matlab Component Runtime)来执行。新的 Matlab 编译器架构形成了新的 Matlab 与 C/C++混合程序设计特点：

- Matlab 程序在 MCR 环境下与在 Matlab 环境下执行的效率相同，因此，通过编译 Matlab 程序不会提高 Matlab 程序的效率。
- MCR 的启动时间与 Matlab 程序的启动时间相同，在混合程序设计中应考虑这一点。
- 在 C/C++程序中无论采用哪种调用方式调用 Matlab 程序，最终结果都是由 MCR 执行 Matlab 程序。
- 由于 Matlab 编译器只生成 Matlab 接口文件，而 MCR 接口一般会采用兼容设计，因此，与 Matlab 6.5 及以前版本的软件相比，用户开发 C/C++混合程序设计的复杂度降低，可继承性提高。
- 由于采用 MCR 执行 Matlab 程序而不是将 Matlab 程序编译为 C/C++程序，Matlab 函数和工具箱中可编译的部分大大增加，为用户开发混合编程提供了方便。

正是存在这些诸多不同，原有的 C/C++与 Matlab 混合程序设计的具体实现方法需要进行修正。为此，笔者对《精通 Matlab 与 C/C++混合程序设计》第 1 版中的内容进行了相应增删或修正，形成了本书的内容。

所谓“万变不离其宗”，虽然 Matlab 编译器的架构发生了很大的变化，但 Matlab与 C/C++混合程序设计还是继承了原有思路。读者在应用中可根据自己的需求选择 Matlab 调用 C/C++程序(MEX 文件)、将 Matlab 程序编译为独立可执行文件、C/C++程序调用 Matlab 引擎、C/C++程序调用 Matlab 程序编译后的动态链接库以及 C/C++程序调用 Matlab 程序编译后的 COM 组件等方式进行。

Matlab 调用 C/C++程序通过将其编译为 MEX 文件来实现，Matlab 提供了一组 C 语言 API 函数供用户调用。这组 API 函数是 Matlab 与用户 C 程序之间的桥梁。通过调用 C/C++程序编译的 MEX 文件，用户可以将 Matlab 程序中运算效率不高的代码用 C/C++来实现，从而提高计算效率。

C/C++调用 Matlab 程序用户可以选择调用 Matlab 程序编译后的动态链接

库或 Matlab 程序编译后的 COM 组件,这两种调用方法的执行效率是相同的。动态链接库方式实现起来比较简单,COM 组件方式实现起来较复杂。除此之外,C/C++可以通过 Matlab 引擎直接执行 Matlab 程序,但这种方式不能脱离 Matlab 环境执行。

另外,书中还介绍了另外两种 Matlab 与 C/C++混合程序设计的方法:Matcom C/C++数学库和 Matlab C++数学库。其中,Matcom 是第一个可以将 Matlab *.m 文件编译为 C/C++代码的编译器。Matcom 可以直接将 m 文件编译为 C/C++代码,但只支持Matlab 5.3版。现在一般情况下没有必要使用 Matcom编译 Matlab 程序,但 Matcom 的 C++矩阵库仍然有一定的使用价值。Matlab C++数学库是 Matlab 提供的一组封装好的矩阵运算数学库,其使用方法和 Matlab 环境中的编写方法十分类似。如果用户用 Visual C++实现用户界面,而又希望寻找一组高效的矩阵运算数学库的话,那么 Matlab C++数学库是一个不错的选择。

Matlab 与 C/C++混合程序设计方法各有千秋,具体应用还要结合开发者的具体情况进行选择。但无论使用哪种方法,Matlab 的数据结构与 C/C++的数据结构之间的相互访问和转换都是混合编程的关键,这也是本书重点所在,希望读者在阅读和开发过程中引起注意。

本书所有的源代码均可在附带的光盘中找到。第 7 章"Matcom 与 C/C++"的开发和编译环境为 Visual C++ 6.0 与 Matcom 4.5.1;第 8 章"VC++调用 Matlab C++数学库"的开发和编译环境为 Visual C++ 6.0 与 Matlab 6.5.1;其他各章的开发和编译环境为 Visual C++ 6.0与 Matlab 2007。

在本书的编写过程中有幸得到很多同志的支持和帮助,在此感谢所有为本书的完成提供过帮助的同事和朋友。感谢网络上提供 Matlab 与 C/C++混合程序设计资料的网友们,在学习 Matlab 与 C/C++混合程序设计的过程中,这些资料使我受益匪浅。感谢我的妻子齐春溪女士,在她的大力支持和协助之下此书方得以顺利编写完成。

由于作者水平有限,对于书中存在的疏漏之处,请不吝赐教,并欢迎读者与作者探讨交流,电子邮件可发送到 matlab_vc_program@yahoo.com.cn,谢谢!

作　者

2007.10.20

目　　录



第1章　Matlab 程序设计初步

1.1　Matlab 程序设计特点

Matlab 语言是一种解释型的高级语言，它包含自己的数据结构、程序流控制及文件输入输出等功能。Matlab 语句可以在 Matlab 控制窗口中直接执行，也可以采用脚本(script) *.m 文件和函数(function) *.m 文件的形式来实现。

1.1.1　Matlab Script 文件

采用 Matlab Script 文件，可以一次执行多条 Matlab 语句，这是一种比较简单的实现方式。由于 Matlab 的变量不需要定义，所以，可以很方便地按照程序的计算流程编写 Matlab Script 文件。如下面的 testscript.m 文件就是一个采用 Script 文件一次执行多条 Matlab 语句的实例。

```
%保存为 testscript.m 文件
x=-pi:0.01:pi;                                   %pi 在 Matlab 中是常量,代表圆周率
y=sin(x);                                        %生成绘制数据
ynoise=y+(rand(1,length(y))-0.5)*0.2;            %加均匀噪声
plot(x,y,'r.');
hold on;                                         %重复绘制不擦除背景
plot(x,ynoise);
hold off;
```

上述 Script 文件的执行结果如图 1-1 所示。

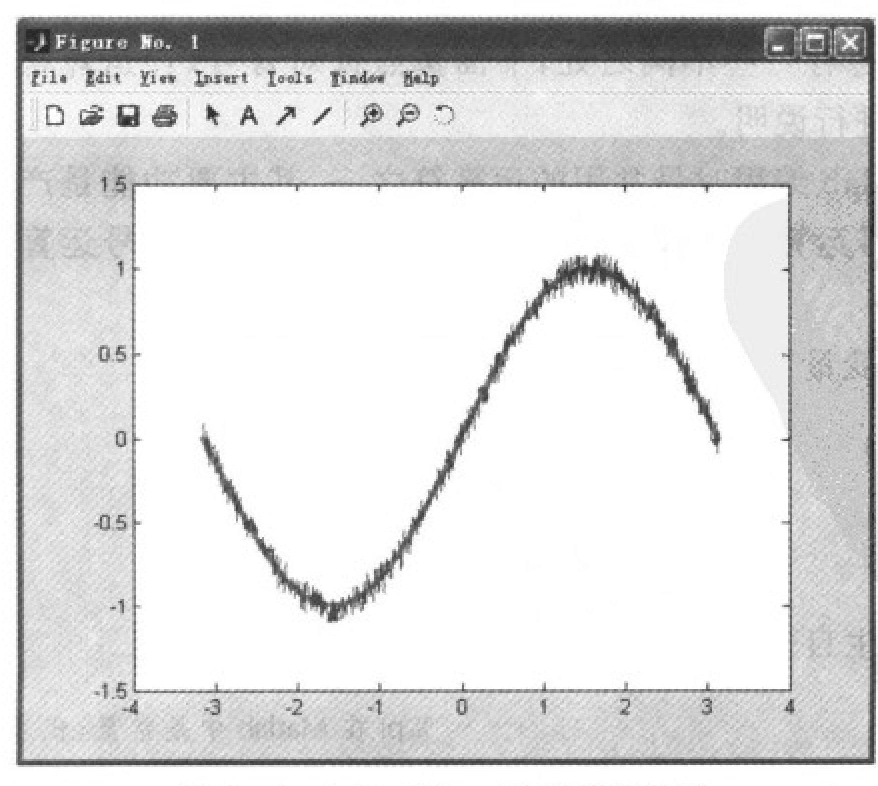


图 1-1　testscript.m 文件执行结果

1.1.2 Matlab 表达式

Matlab 的主要运算符如表 1-1 所列。

表 1-1 Matlab 的主要运算符

算术运算符		关系运算符		逻辑运算符	
+	数值阵列元素加	<	小于	&	按位与
−	数值阵列元素减	<=	小于等于	|	按位或
.*	数值阵列元素乘	>	大于	~	按位非
./	数值阵列元素右除	>=	大于等于	xor	异或
.\	数值阵列元素左除	==	相等	&&	逻辑与
:	冒号运算符	~=	不相等	||	逻辑或
.^	数值阵列元素幂				
.'	数值阵列元素转置				
'	数值阵列转置(对于实数与.'相同;对于复数,转置的同时对复数求共轭)				
*	数值阵列乘				
/	数值阵列右除				
\	数值阵列左除				
^	数值阵列幂				

Matlab 的算术运算符对于矩阵运算是非常方便的,可以大致将其分为针对数值阵列元素和针对数值阵列整体的两类数学运算符。其中针对数值阵列元素的数学运算符的运算方式可以理解为是数值阵列的单个数学元素逐个按顺序进行运算的运算符,而针对数值阵列整体的数学运算符的运算对象则是数值阵列整体(一般情况下为矩阵和向量)。Matlab 的运算符与普通高级语言的运算符有一些不同之处,下面重点针对冒号(:)运算符、点(.)运算符及左除(\)和右除(/)运算符进行说明。

":"运算符是 Matlab 编程时最常用的运算符之一,其主要功能是产生一个等间距的数据向量。恰当地使用冒号运算符可以使 Matlab 程序非常简洁。冒号运算符最常见的两个用途如下所述。

① 产生循环控制变量

```
for i=1:100
    %循环程序
end
```

② 生成数据时产生自变量

```
x=-pi:0.01:pi;                    %pi 在 Matlab 中是常量,代表圆周率
y=sin(x);
```

“.”运算符和其他运算符联合使用，则表示对阵列的元素进行操作。如 A. * B 表示 A 和 B 的相同位置元素进行相乘，A * B 则表示阵列 A 与 B 相乘(矩阵相乘或者向量相乘)。

“/”和“\”分别表示 Matlab 数值阵列右除和左除，其中 A/B 表示线性方程 AX=B 的解，A\B 表示线性方程 XA=B 的解；如果“/”或“\”与“.”配合使用，则表示数值阵列相同位置的元素分别进行右除或左除，其中 A.\B 表示 B(i)/A(i)，A./B 表示 A(i)/B(i)。

下面的一组 Matlab 语句则说明了 Matlab 算术运算符的使用方法。

```
%保存为 testoperat.m 文件
A=[1 2 3;4 5 6;7 8 9;];
B=[1 1 1;1 1 1;1 1 1;];

%矩阵元素相加、相减、相乘
C=A+B;
D=A-B;
E=A.*B;

%矩阵元素左除和右除
%其中 F 和 G 的结果相同
%均为：
%     1.0000     0.5000     0.3333
%     0.2500     0.2000     0.1667
%     0.1429     0.1250     0.1111
F=A.\B;
G=B./A;

%冒号运算符
N=1:100;

%幂运算符
A1=A.^2;

%转置
%Z1 为：
%       1.0000+1.0000i    4.0000+1.0000i    7.0000+1.0000i
%       2.0000+1.0000i    5.0000+1.0000i    8.0000+1.0000i
%       3.0000+1.0000i    6.0000+1.0000i    9.0000+1.0000i
%Z2 为：
%       1.0000-1.0000i    4.0000-1.0000i    7.0000-1.0000i
%       2.0000-1.0000i    5.0000-1.0000i    8.0000-1.0000i
%       3.0000-1.0000i    6.0000-1.0000i    9.0000-1.0000i
Z=A+B*i;                  %构造复数
Z1=Z.';
Z2=Z';
```

```
A=[1 1 1;2 1 2;1 2 3;];
B=[1 1 1]';

%计算方程 AX=B 的解
%x+y+z=1
%2x+y+2z=1
%x+2y+3z=1
X=A\B;                  %X=[0.5000 1.0000-0.5000]'

%计算方程 XA=B' 的解
%x+2y+z=1
%x+y+2z=1
%x+2y+3z=1
X=B'/A;                 %X=[1 0 0]
```

Matlab 的关系运算符和逻辑运算符与 C 语言的使用方法一样，只是 C 语言中表示非的“!”运算符在 Matlab 中用“～”来代替。另外，Matlab 的逻辑运算符可以直接对数值阵列进行操作，例如：

```
A=[1 1 0 0 1];
B=[0 1 0 0 1];
C=A&B;                  %C=[0 1 0 0 1]
D=A|B;                  %D=[1 1 0 0 1]
E=~A;                   %E=[0 0 1 1 0]
F=xor(A,B);             %F=[1 0 0 0 0]
```

1.1.3　Matlab 函数

除了采用 Script 文件编写 Matlab 程序以外，Matlab 语言也可以通过函数的形式编写 Matlab 程序。通过 Matlab 函数编写的 Matlab 程序便于维护，而且对于使用者而言，只需要了解函数的输入和输出即可，因此，要比 Script 文件容易使用得多。

Matlab 函数的定义方法如图 1-2 所示。

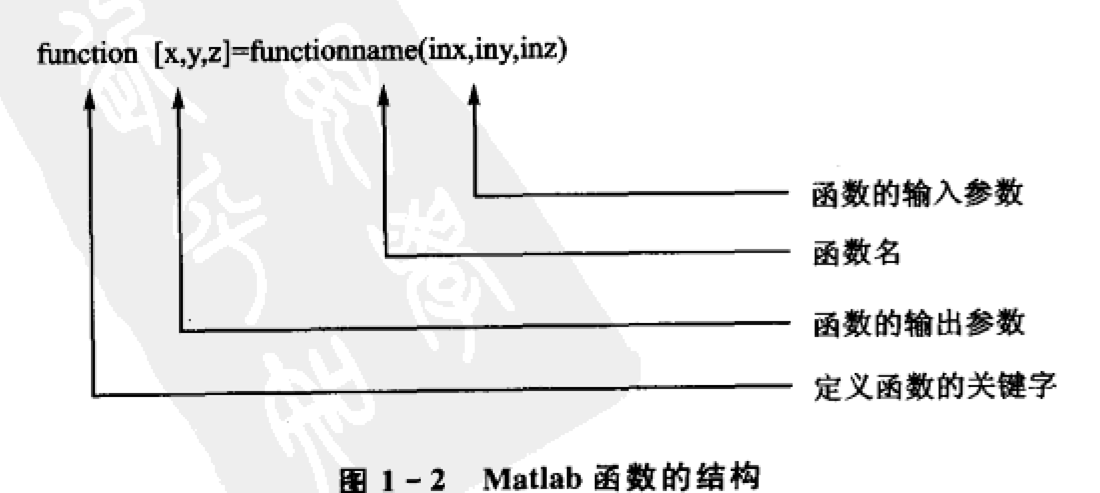


图 1-2　Matlab 函数的结构

函数体的写法与 Script 文件的写法类似，使用函数时需要注意的是：第一，保存 Matlab 函数程序时必须采用函数名作为文件名，其扩展名为. m；第二，Matlab 函数输入的外部变量如果在函数体内发生变化，则函数结束时，外部变量的值不会发生变化，类似于 C 语言输入参数的值传递。

如编写下面的函数，试图调换 x 和 y 的值，实际上这样做是不能实现的。

```
%保存为 change.m 文件
function [] = change(x,y)
t=x;
x=y;
y=t;
```

测试程序如下：

```
x=10;
y=1;
change(x,y);            %此时 x=10,y=1
```

另外，Matlab 有两个函数 nargin 和 nargout，分别返回 Matlab 函数的输入和输出参数的个数。如果采用元组变量 varargin 和 varargout 来替代常规的输入和输出参数，则函数就可以接受数目可变的输入和输出参数。下面的例子就是用来说明这两个函数的用法。

```
%保存为 testnarin_out.m 文件
function [varargout] = testnarin_out(varargin)
% function [x,y,z] = testnarin_out(m,n,k)
disp('输入参数的个数');
disp(nargin);
disp('输出参数的个数');
disp(nargout);
if(nargout>nargin)
    error('输出参数的个数不能大于输入参数的个数!\n');
end
varargout{1:nargout} = varargin{1:nargout};
```

测试程序如下：

```
m=rand(1,10);n=randn(5);z=magic(4);
testnarin_out(m,n);
x=testnarin_out(m,n);
[x,y,z] = testnarin_out(m,n);
```

则程序运行结果如下所示。

```
输入参数的个数
2
输出参数的个数
0
```

```
输入参数的个数
2
输出参数的个数
1
输入参数的个数
2
输出参数的个数
3
??? Error using==> testnarin_out
输出参数的个数不能大于输入参数的个数!\n
```

1.1.4 Matlab 的向量运算

Matlab 程序设计语言是解释型语言,其循环语句的效率非常低。但是,由于在算法上作了特殊的优化,Matlab 程序设计语言对于向量和矩阵运算的效率却很高。因而在进行 Matlab 程序设计时,一个很重要的原则就是尽量少地使用循环语句。可以做一个测试,同样是乘法,对于采用 for 循环和阵列元素相乘运算符“.*”,两者的执行时间可以相差数十倍。

```
%保存为 testfor.m 文件
%清空 workspace 的变量
clear all;
clc;
%Matlab 循环语句与向量运算的测试语句
a = 1:1000000;
b = 1000000:-1:1;
sum_axb = 0 ;
tic;                                  %计时开始
for i = 1:1000000
    sum_axb = sum_axb + a(i) * b(i);
end;
time1 = toc;                          %计时结束并输出采用循环语句进行运算的时间

sum_axb = 0;
tic;                                  %计时开始
sum_axb = sum(a. * b);
time2 = toc;                          %计时结束并输出采用向量运算消耗的时间

result1 = strcat('采用循环语句运算消耗的时间为:',num2str(time1),'秒');
result2 = strcat('采用向量运算消耗的时间为:      ',num2str(time2),'秒');

disp(result1);
disp(result2);
```

测试结果为：

采用循环语句运算消耗的时间为：0.718 秒
采用向量运算消耗的时间为：　0.031 秒

将 Matlab 的循环语句运算转换为向量运算需要在 Matlab 程序编写过程中加入一些技巧，其中最常见的形式如下所述。

① 用向量化运算代替 for 和 while 循环运算。如：

```
for i = -pi:0.1:pi
    y(i) = sin(i);
end
```

可以用下面的向量运算来代替。

```
x = -pi:0.1:pi;
y = sin(x);
```

另外，如果上例中还有两个向量之间的运算，可以用类似“.*”的数值阵列元素的运算函数来代替。即

```
for i = 1:1000000
    sum_axb = sum_axb + a(i) * b(i);
end;
```

可以被

```
sum_axb = sum(a.*b);
```

来代替。

② 采用一些经过优化的 Matlab 向量运算函数。

经常用于向量运算的 Matlab 函数如表 1-2 所列。

表 1-2　Matlab 常用向量函数

函数名	函数功能
all	判断数值阵列是否全部非零
any	判断数值阵列是否有非零元素
reshape	变换数值阵列的各维元素数目
find	返回非零元素在阵列中的位置及其数值
sort	将数组按升序排序
sum	求数组的和
repmat	扩展阵列

除了 repmat 函数，表 1-2 中的其他函数都比较容易理解。下面就以 repmat 函数为例，说明如何采用 Matlab 的向量函数替换循环语句的技巧。repmat 函数用于扩展 Matlab 阵列，对于矩阵

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

它的扩展矩阵 B 为

$$B = \text{repmat}(A,5,3) = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \end{bmatrix}$$

下段程序运用 repmat 函数绘制如图 1-3 所示的三维曲面图。

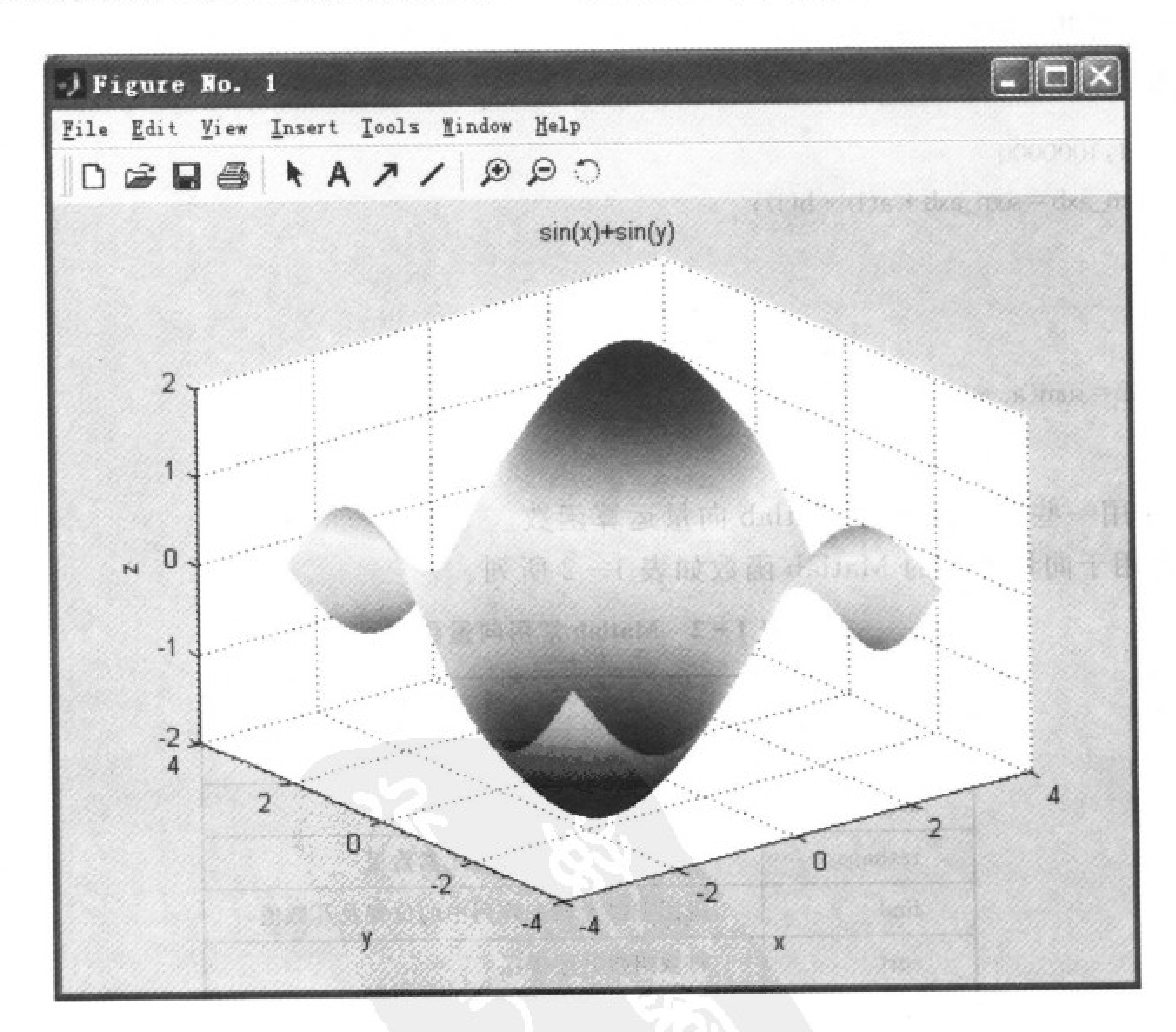


图 1-3　sin(x)+sin(y)的三维曲面图

```
%保存为 testrepmat.m 文件
%运用 repmat 函数
%绘制 sin(x)+sin(y)的三维曲面图
%首先清空 workspace 的变量
```

```
clear all;
clc;
x = -pi:0.01:pi;
y = -pi:0.01:pi;
y = y';
x1 = repmat(x,length(x),1);
y1 = repmat(y,1,length(y));
z = sin(x1) + sin(y1);
mesh(x1,y1,z);
xlabel('x');
ylabel('y');
zlabel('z');
title('sin(x) + sin(y)');
```

其实，采用向量运算代替循环语句的本质是用牺牲内存空间的做法来换取计算效率的提高。这是因为所有 Matlab 的内置函数都针对向量和矩阵运算进行了优化，因而对于 Matlab 的内置函数来说，只有当输入的数据是向量或者矩阵的形式时，其计算效率才是最高的。为了提高计算效率，就必须满足 Matlab 内置函数的这种要求，即在调用 Matlab 函数以前通过适当技巧构造好其所需要的向量或者矩阵，这便是 Matlab 程序设计向量化的本质所在。

1.1.5 Matlab 的程序控制

Matlab 用于程序控制的关键字如表 1-3 所列。

表 1-3　Matlab 常用的程序控制关键字

关键字	关键字的功能描述
if	与 else 和 elseif 关键字联合使用，根据设定的逻辑条件执行不同的代码
switch	与 else 和 otherwise 关键字联合使用，根据设定的变量值的不同执行不同的代码
while	根据设定的逻辑条件执行循环次数不定的循环模块
for	执行循环次数一定的循环模块
continue	适用于 for 或者 while 循环中，中止当前循环体语句的执行，直接进行下一次循环
break	退出当前层的循环体
return	直接返回到当前函数的调用函数中

下面分别举例说明 Matlab 程序控制关键字的使用方式。

1. if 和 switch 语句

if 和 switch 语句是 Matlab 程序设计中选择程序结构的两个关键字。其中 if 语句根据设定的逻辑条件执行不同的代码，而 switch 语句则根据设定的变量值的不同执行不同的代码。

if 可以单独使用，也可以与 else 和 elseif 联合使用，其使用方式有下面 3 种。

(1) 单独使用

```
if 逻辑表达式
    语句
```

```
end
```

例如：

```
if A>B
    disp('A 大于 B');
end
```

(2) 和 else 联合使用

```
if 逻辑表达式
    语句 1
else
    语句 2
end
```

例如：

```
if A>B
    disp('A 大于 B');
else
    disp('B 大于等于 A');
end
```

(3) 和 elseif 联合使用

```
if 逻辑表达式
    语句 1
elseif
    语句 2
else
    语句 3
end
```

例如：

```
if A>B
    disp('A 大于 B');
else if A==B
    disp('A 等于 B');
else
    disp('B 大于 A');
end
```

而 switch 语句的使用方式如下。

```
switch 表达式
    case 数值 1
        语句 1                %当表达式的值为数值 1 时执行语句 1
```

```
    case 数值 2
        语句 2              %当表达式的值为数值 2 时执行语句 2
    ⋮
    otherwise
        语句 n+1            %当表达式的值与数值 1～数值 n 皆不相等时执行此语句
end
```

与 C 语言类似，switch 语句与 if 语句都是程序设计中选择程序结构的关键词，但是两者的应用场合略有不同。下面，是一个判断输入变量符号的小例子，并分别用 if 语句和 switch 语句来实现，通过此例，读者可以大致了解 if 和 switch 语句在使用上的不同特点。

```
%保存为 testif.m 文件
%if 语句的使用方法测试
function [] = showInputSign_if(input)
%根据输入数据的正负不同，给出不同的输出
nlen = length(input)；
if nlen>1
    disp('输入的数据不是 1×1 的数值阵列！')；
else
    if input ～=0                 %注意"～="表示"不等于"
        if input > 0
            disp('输入数据大于 0')；
        else
            disp('输入数据小于 0')；
        end
    else
        disp('输入数据等于 0')；
    end
end

%switch 语句的使用方法测试
function [] = showInputSign_switch(input)
%根据输入数据的正负不同，给出不同的输出
nlen = length(input)；
if nlen>1
    disp('输入的数据不是 1×1 的数值阵列！')；
else
    signinput = sign(input)；
    switch signinput
        case 0
            disp('输入数据等于 0')；
        case 1
```

```
            disp('输入数据大于0');
        case -1
            disp('输入数据小于0');
        otherwise
            disp('这是不可能的!!!!!!! ');
    end
end
```

2. for 和 while 语句

for 和 while 语句是 Matlab 程序设计中的两个循环关键词,其中 for 循环用于执行指定次数的 Matlab 循环语句,while 循环用于执行不定次数的 Matlab 循环语句。for 循环语句和 while 循环语句的使用方法都比较简单,如下所示。

(1) for 循环语句使用方式

```
for 索引 = 开始:步长 :结束
    语句
end
```

例如:

```
for i=3:2:9
    x(i)=2*x(i-2);
end
```

(2) while 循环语句使用方式

```
while 逻辑表达式
    语句
end
```

例如:

```
sum=0;
while x(i)>0
    sum=sum+x(i);
end
```

在 for 和 while 循环结构中需要使用 break 和 continue 这两个关键字,其中 break 关键字表示退出当前循环,而 continue 表示结束当前循环语句执行,进入下一次循环。这与 C 语言的使用方法类似。

1.2 Matlab 常用的数据类型

Matlab 所有的数据类型都可以用一种数据类型即 Matlab 阵列(Array)来表达。常用的 Matlab 阵列类型有数值类型(整型、单精度型和双精度型)(numeric array)、元组类型(cell array)、结构体类型(structure array)、字符类型(char array)和逻辑型(logical array)。另外还

有其他一些类型的 Matlab 阵列，如图 1－4 所示。

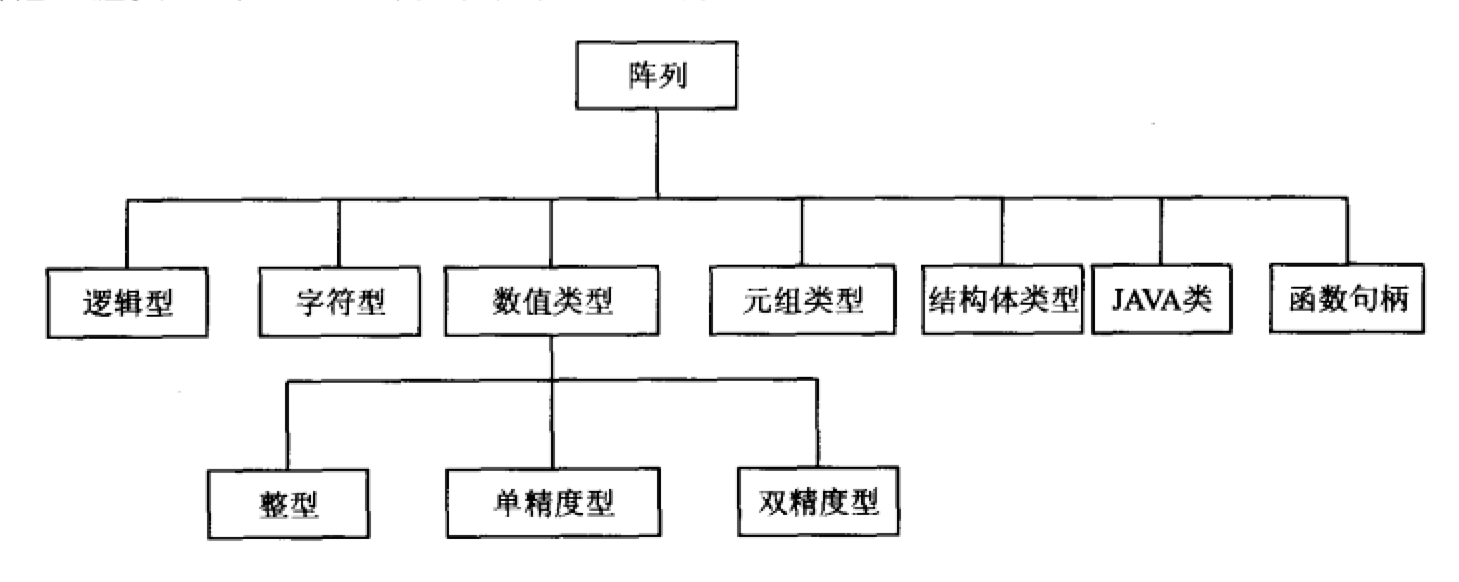


图 1－4　Matlab 阵列数据类型

广义上说，Matlab 只有一种数据类型，即阵列，只不过 Matlab 阵列有不同的类型。Matlab 阵列类似于数组的概念，一个 Matlab 阵列可以看做是某一 Matlab 数据类型（狭义的）的数组。举例来说，一个 m×n 的结构体阵列类似于一个 m×n 的结构体数组，一个 m×n 的双精度型阵列类似于一个 m×n 的双精度型数据的数组。如果 m＝1，n＝1，则此时的 1×1 阵列相当于 C 语言的变量。对于多维数组，与 C 语言的数组元素在内存中按行排列的排列顺序不同，所有 Matlab 阵列在内存中都是按列排列的。

各种类型的阵列在 Matlab 中使用之前是不用声明的，而且各种类型阵列的使用方式十分相似。但是如果进行 Matlab 与 C/C++混合编程，Matlab 数据类型和 C/C++数据类型的交互是需要解决的关键问题之一。因而，熟悉和掌握 Matlab 数据类型的特点对于 Matlab 与 C/C++ 混合编程的实现是非常重要的。下面，就 Matlab 几种主要阵列类型的使用进行说明。

1.2.1　数值阵列

Matlab 的主要核心部分就是高效率的数值计算，因而数值类型的 Matlab 阵列也是 Matlab 开发环境中最常用的阵列类型。Matlab 的数据类型大致有整型、单精度浮点型和双精度浮点型 3 类，其中整型又有 8 位、16 位、32 位、64 位及无符号和有符号之分。由于 Matlab 所有的运算都是采用双精度浮点型，因而整型阵列如果要参加运算，必须先转换为双精度型。同样由于 Matlab 所有的运算都是采用双精度浮点型，因而，Matlab 数值阵列中最常用和最方便的是双精度浮点型阵列。Matlab 数值阵列的初始值可以直接设定，也可以通过其他函数生成。例如：

```
A=[1 2 3;4 5 6;7 8 9;];          %直接设定双精度数值阵列 A 的值
B=rand(3,3);                     %通过函数生成双精度阵列 B 的值
```

其中，“[”、“]”和“;”是 Matlab 比较常用的符号。“[”和“]”表示 Matlab 阵列构造的开始和结束；“;”在“[”和“]”之间表示 Matlab 阵列一行的结束，如果“;”放在一条语句的末尾，则表示本条语句的输出不在 Matlab Command 窗口中显示，否则要在 Matlab 的 Command 窗口中显示。比较下面两条语句在 Matlab Command 窗口中执行结果的不同之处：

```
>> C=magic(3)
C=
     8     1     6
     3     5     7
     4     9     2
>> C=magic(3);
>>
```

实际上，Matlab 数值阵列与 C/C++中数组的概念类似，只不过在存储方式上，Matlab 与 FORTRAN 相同，多维数组采用按列存储的方式，而 C/C++则采用按行存储，在进行 Matlab 与 C/C++混合编程时，这是一个重要的细节。

下面，给出一个利用 Matlab 读取灰度位图的数据，并进行二值化的例子。通过这个实例将会熟悉 Matlab 数值阵列的使用。

```
%保存为 processgrayimage.m
function []=processgrayimage()
% function []=processgrayimage()
%说明：
%       Matlab 读取的位图图像数据是 8 位无符号整型
%       Matlab 显示和存储图像时，也需要是 8 位无符号整型
%       或者将所有的数据归一到[0 1]之间
%       因而采用 double 和 uint8 进行整型和双精度型之间的转换就比较方便
[name,path]=uigetfile({'*.bmp','请选择一个位图文件(*.bmp)'},'请打开一个位图文件');
file=strcat(path,name);
[I,map]=imread(file);
if size(I,3)==3
   I=rgb2gray(I);
end
%将图像数据转换为 double 型数据以方便处理
I=double(I);
I1=I-100;
signI1=sign(I1);
coefI1=(signI1+abs(signI1))/2;
%大于 125 的图像部分
I1=I.*coefI1;
%小于 125 的图像部分
I2=I.*(1-coefI1);
I1=(I1/max(max(I1)))*255;
I2=(I2/max(max(I2)))*255;
%将数据转换为 unsigned int8 型数据，以方便进行显示
I1=uint8(I1);
I2=uint8(I2);
figure;
```

```
h1 = subplot(1,2,1);
subimage(I1);
h2 = subplot(1,2,2);
subimage(I2);
truesize;
```

程序中要求打开一个位图文件的对话框如图 1－5 所示，程序执行的结果如图 1－6 所示。

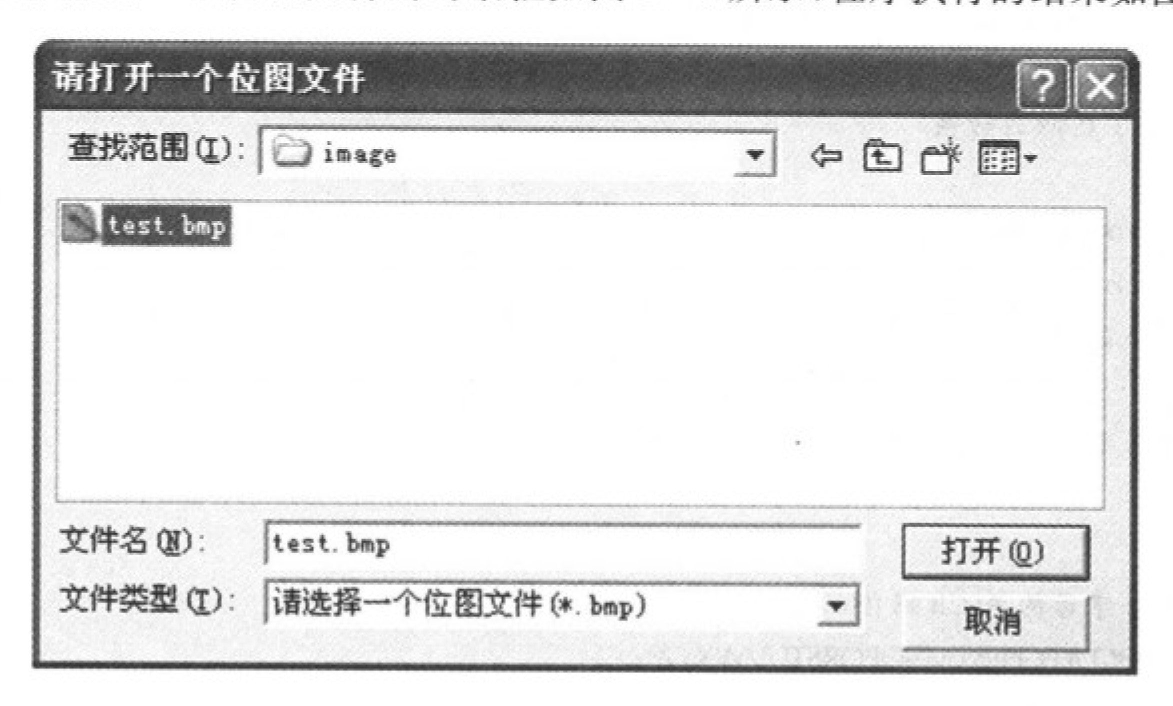


图 1－5　选择要处理的文件

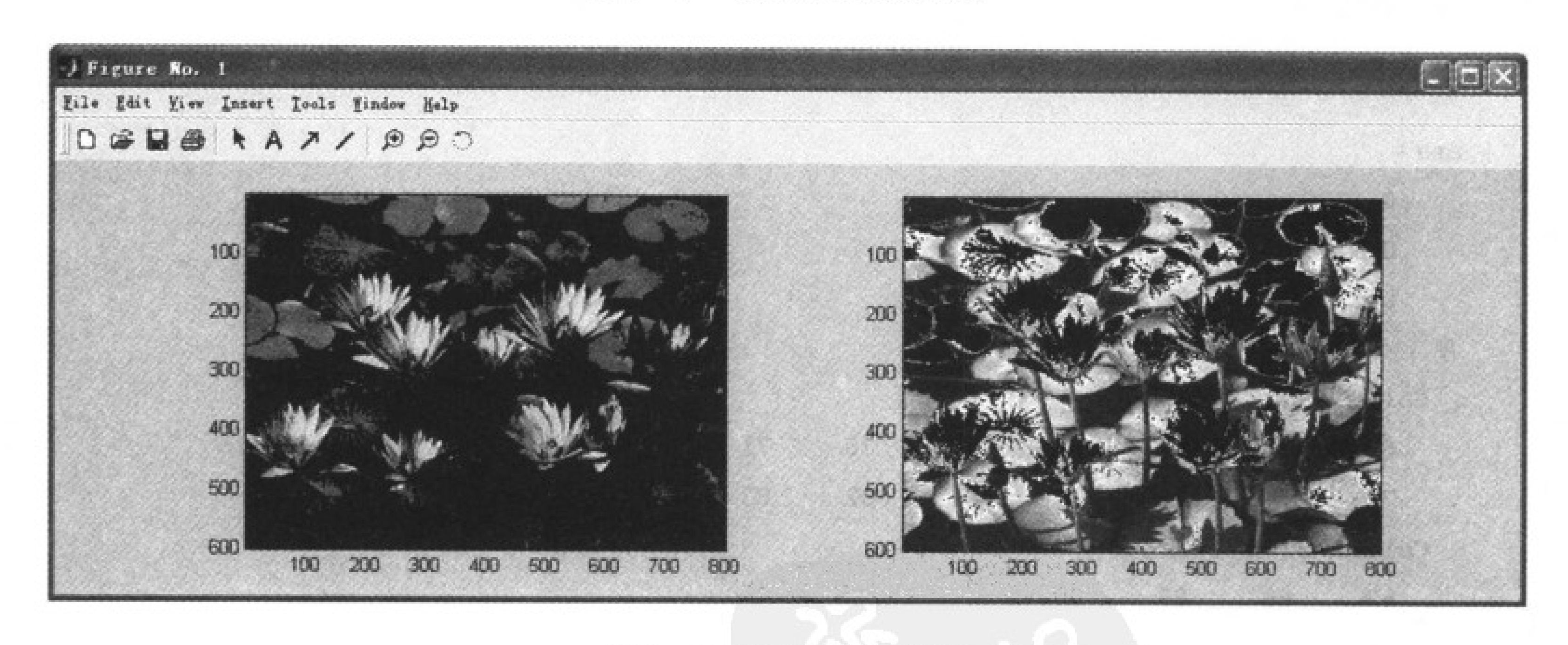


图 1－6　程序执行的结果

1.2.2　字符阵列

和 C/C++不同，Matlab 的字符采用 UNICODE 标准，用双字节表示。Matlab 字符阵列可转换为相应的 ASCII 码阵列，普通数值阵列也可以转换为字符型阵列。Matlab 提供了常用的字符串操作函数，下面通过实例可以熟悉 Matlab 的字符串操作。

```
%保存为 teststring.m 文件
function [] = teststring()
```

```
%Matlab 字符操作函数
%mat2str 函数生成 eval 可以执行的字符串
data = rand(3,3);
sdata = mat2str(data);
eval(sdata);

%掷硬币的游戏,如果 is 是"1"则为正面,否则为反面
%字符串比较
%字符和数字之间的转换
str = '1';
is = num2str(round(rand));
if strcmp(str,is)
  disp('硬币的正面');
else
  disp('硬币的反面');
end

%输出 26 个字母的 ASCII 码值
strLetter = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
valLetter = double(strLetter);
disp(valLetter);
```

程序运行结果如下所示。

```
ans =
    0.1730    0.2523    0.1365
    0.9797    0.8757    0.0118
    0.2714    0.7373    0.8939
硬币的反面
  Columns 1 through 22
    65    66    67    68    69    70    71    72    73    74    75    76
    77    78    79    80    81    82    83    84    85    86
  Columns 23 through 26
    87    88    89    90
```

1.2.3　元组阵列

Matlab 元组是 Matlab 特有的数据结构,它是一种特殊的阵列。Matlab 元组的元素可以是任意一种 Matlab 类型的阵列。如图 1-7 所示是一个 2×3 的元组阵列,包括结构体阵列、数值型阵列、字符型阵列及元组阵列,它们都可以作为元组阵列的一个元素。

元组阵列的索引方式与数值型阵列、结构体阵列及字符型阵列不同,可以采用"{"和"}"及"("和")"两种索引方式。这两种方式的不同之处是通过"{"和"}"得到的是相应的元组阵列元素,而通过"("和")"得到的却是一个包含相应元组阵列元素的 1×1 元组阵列。下面的实例给出了图 1-7 中所示元组阵列的创建过程。

cell(1,1)	cell(1,2)	cell(1,3)
$\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$	name: '李月' college: '北京科技大学' phone: '01063320501' specialties: { '平面设计', '网站制作' }	$\begin{bmatrix}2+3i & 5+6i\\3+3i & 0.1+0.558i\end{bmatrix}$
cell(2,1)	**cell(2,2)**	**cell(2,3)**
'string' ; [2 4; 6 8] ; [7 8 9] ; 32+i	'Hello World!'	$\begin{bmatrix}9&8&6\\5&4&3\\2&1&7\end{bmatrix}$

图 1-7　元组结构示意图

```
%保存为 testcell.m 文件
function []=testcell()
% function []=testcell()
tcell=cell(2,3);
tcell{1,1}=[1 2 3;4 5 6;7 8 9;];
%... 表示续行符号
tcell{1,2}=struct('name','李月',...
          'college','北京科技大学',...
          'phone','01063320501',...
          'specialties',{'平面设计','网站制作'});
tcell{1,3}=[2+3*i 5+6*i;3+3*i 0.1+0.558*i;];
tempcell=cell(2,2);
tempcell{1,1}='string';
tempcell{1,2}=[2 4;6 8;];
tempcell{2,1}=[7 8 9];
tempcell{2,2}=32+i;
tcell{2,1}=tempcell;
tcell{2,2}='Hello World! ';
```

```
tcell{2,3}=[9 8 6;5 4 3;2 1 7;];
%采用"{"和"}"索引得到相应元组阵列元素
for i=1:2
    for j=1:3
        disp(tcell{i,j})
    end
end
%通过"("和")"索引得到 1×1 的相应元组阵列
for i=1:2
    for j=1:3
        disp(tcell(i,j))
    end
end
```

程序运行结果如下所示。

```
1    2    3
4    5    6
7    8    9
1×2 struct array with fields:
    name
    college
    phone
    specialties
   2.0000 + 3.0000i    5.0000 + 6.0000i
   3.0000 + 3.0000i    0.1000 + 0.5580i
    'string'               [2×2 double]
    [1×3 double]    [32.0000 +  1.0000i]
Hello World!
     9    8    6
     5    4    3
     2    1    7
    [3×3 double]
    [1×2 struct]
    [2×2 double]
    {2×2 cell}
    'Hello World! '
    [3×3 double]
```

1.2.4 结构体阵列

Matlab 结构体与 C/C++语言结构体类似,结构体由不同的域(field)组成,其中每个域由域名和域值组成。Matlab 结构体阵列是 Matlab 结构类型的数组,Matlab 结构体阵列的结构示意图如图 1-8 所示。

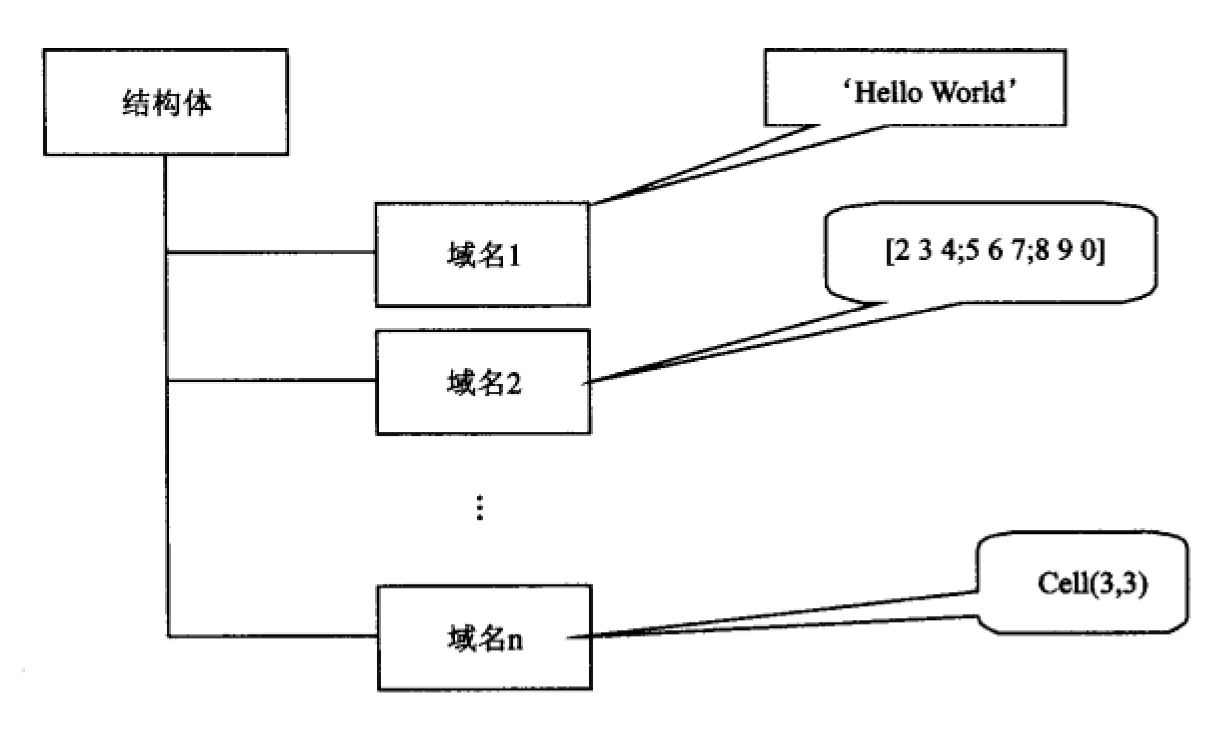


图 1－8　结构体示意图

下面，通过一个实例来说明结构体的构成和基本操作方法。

```
%保存为 teststruct.m
function [] = teststruct()
%构造个人通讯录结构体
%个人情况：
%姓名 name
%地址 addr
%联系电话 phone
%爱好 hobby
info(1).name = '李芳';
info(1).addr = '南京市南京大学数学系 3#335';
info(1).phone = '02552528888';
info(1).hobby = {'旅游','音乐','书法'};
info(2).name = '欧阳';
info(2).addr = '哈尔滨市哈尔滨工业大学计算机系 99-335';
info(2).phone = '045182589666';
info(2).hobby = {'篮球','游泳','游戏'};
disp(info(1));
disp(info(2));
%删除域
info = rmfield(info,'hobby');
disp(info(1));
disp(info(2));
```

程序运行结果如下所示。

```
info =
      name: '李芳'
      addr: '南京市南京大学数学系 3#335'
```

```
      phone: '02552528888'
info =
1 × 2 struct array with fields:
     name
     addr
     phone
     hobby
     name: '李芳'
     addr: '南京市南京大学数学系 3#335'
     phone: '02552528888'
     hobby: {'旅游'  '音乐'  '书法'}
     name: '欧阳'
     addr: '哈尔滨市哈尔滨工业大学计算机系 99-335'
     phone: '045182589666'
     hobby: {'篮球'  '游泳'  '游戏'}
     name: '李芳'
     addr: '南京市南京大学数学系 3#335'
     phone: '02552528888'
     name: '欧阳'
     addr: '哈尔滨市哈尔滨工业大学计算机系 99-335'
     phone: '045182589666'
```



第 2 章　Matlab 编译器

2.1　Matlab 编译器技术概述

Matlab 编译器技术的基本架构如图 2－1 所示。Matlab 编译器的运行机制可作如下描述：

- Matlab 程序通过 Matlab 编译器编译为可执行文件、动态链接库或者 COM 组件。
- 如果编译为可执行程序，那么当执行时会自动调用 MCR，或者说是将编译后的 Matlab 代码传递给 MCR 来执行。
- 如果编译为动态链接库或者 COM 组件，那么当其被主程序调用时会自动调用 MCR，或者说将编译后的 Matlab 代码传递给 MCR 来执行。

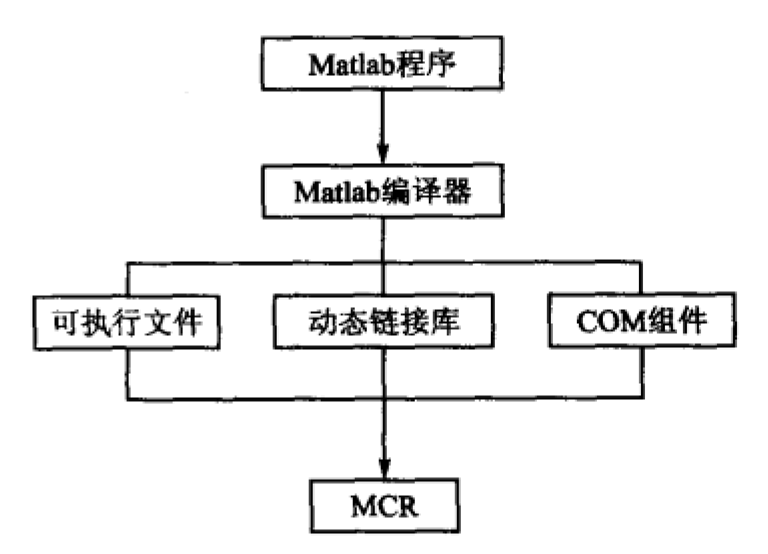


图 2－1　Matlab 编译器技术基本架构

将 Matlab 7.0 以后的编译器与 Matlab 6.5 及其以前的编译器对比，可以发现设计思想上明显的不同之处。Matlab 6.5 及其以前的编译器侧重的是如何将 Matlab 程序编译为 C/C＋＋程序；而 Matlab 7.0 以后的编译器由于采用了 MCR 技术，其侧重只是将 Matlab 程序编译为 MCR 可执行的程序（实际上只是生成接口文件，这在本书后面章节会有详细介绍）。

MCR（Matlab Component Runtime）是 Matlab 提供的一组动态链接库，通过 MCR 可以执行 Matlab 程序。可以把 MCR 想象成一个裁减版的 Matlab，它支持 Matlab 语言的所有特征（包括面向对象编程等）。除了 MCR 以外，Matlab 7.0 以后的编译器还采用了 CTF（Component Technology File）技术，即将最终需要发布的所有程序（包括代码、数据以及其他文件）打包为一个 CTF 文件。为了保证代码的安全性，CTF 还采用了 AES（Advanced Encryption Standard）密码技术进行加密。

2.2 Matlab 编译器的功能

早期的 Matlab 编译器(Matlab 6.5 及其以前)的主要功能有下面几个：

- C/C++和 Matlab 程序互相调用,增强开发效率。
- 加密代码,保护软件开发者的知识产权。
- 加快 Matlab 程序的执行速度。

Matlab 7.0 以后,由于采用 MCR 技术,程序通过 Matlab 和通过 MCR 的执行速度是一致的,因而通过 Matlab 编译器编译 Matlab 程序以后并不会使其执行速度加快(C 编写的 MEX 文件除外)。

在这里,笔者想回顾一下 Matlab 程序编译所经历的大致历程。通过分析该历程可以更加明了为什么需要 Matlab 编译器、在什么情况下选择 Matlab 编译器、应当如何最大限度地利用 Matlab 编译器?

Matlab 出现以后,以其特有的简单便利、强大功能很快拥有了相当多的使用者。但是 Matlab 也存在一些问题,比如循环执行速度慢,开发和发布应用软件不方便等。这时候,人们自然就会想到是不是可以把 Matlab 程序通过编译器编译为 C/C++语言以方便其他程序调用。如图 2-2 所示,这时候 Matlab 编译器需要解决的问题主要有两个：

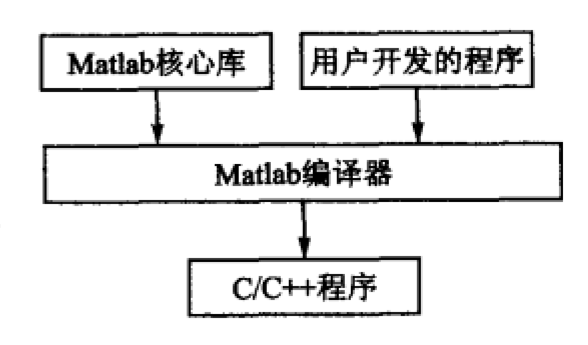


图 2-2 Matlab 编译器的基本功能

① Matlab 核心库调用的接口函数。

② 将用户开发的程序(包括后来开发的各种工具箱)编译为 C/C++程序。

对于第一个问题,查看一下 Matlab 的发展历史,最初 Matlab 的核心矩阵算法库是用 Fortran 编写的,后来使用 C 语言重写。既然是用 C 语言重新编写的,因而在 C/C++程序中对 Matlab 核心库调用的问题应该很容易解决。对于第二个问题,虽然 Matlab 程序从语法上来说非常灵活,程序的规范性较 C/C++程序差很多,但是 Matlab 程序的主要元素实际上非常少,因而编译为 C/C++程序从技术上说也是完全可以解决的。事实的发展也是如此,第三方软件 Matcom 的出现证实了这一点。Matcom 可以将 Matlab 程序完全编译为 C/C++文件(而不是像现在这样只是接口文件),并且提供了一套完整的矩阵运算库。后来,Mathworks 将 Matcom 收购,以此为基础在 Matlab 5.3 中推出了最早的 Matlab 编译器。从 Matlab 5.3 到 Matlab 6.5,尽管 Matlab 编译器基本延续了 Matcom 的设计思想,但在很多方面不断变化,甚至相互之间都存在很大的兼容性问题。有一段时间,令人费解的是,Mathworks 开发的 Matlab 编译器总是变来变去,而且公开的文档少得可怜,很多技术细节都是通过非官方渠道获得的。现在想想,主要的原因有可能还是 Mathworks 出于知识产权保护的考虑。设想一下,如果按照 Matcom 的设计思想,所有的 Matlab 核心库通过动态链接库发布,而且所有的 Matlab 都可以编译为 C/C++程序,那 Mathworks 岂不是把自己所有的核心都拱手公布了吗?说得夸张点,就是人们完全可以再自己开发一个拥有与 Matlab 功能一样强大的软件。Matlab 7.0 以后,Mathworks 通过采用 MCR 就很好地回避了这个问题,而且能够提供一种兼容性较强的 Matlab 编译和混编的方法。当然,Mathworks 或者工具箱开发者对知识产权还

是有些顾虑，所以并不是所有的工具箱都能通过 Matlab 编译器编译。

2.3 使用 Matlab 编译器的准备工作

"工欲善其事，必先利其器"，使用 Matlab 编译器之前需要做一些准备工作，其中包括：

- 安装 Matlab 和 C/C++编译器。Matlab 自带一个 lcc 编译器（只能编译 C 文件），本书一律采用 Visual C++ 6.0。
- 配置 Matlab 编译器。

在命令行运行 mbuild -setup 命令（注意 mbuild 和-setup 之间应有一个空格），然后出现如下信息，根据自己的需求选择相应的选项即可。

```
>> mbuild -setup
Please choose your compiler for building standalone Matlab applications:

Would you like mbuild to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc - win32 C 2.4.1 in D:\Matlab~1\sys\lcc
[2] Microsoft Visual C++ 6.0 in D:\Program Files\Microsoft Visual Studio

[0] None

Compiler: 2

Please verify your choices:

Compiler: Microsoft Visual C++ 6.0
Location: D:\Program Files\Microsoft Visual Studio

Are these correct? ([y]/n): y

Trying to update options file: C:\Documents and Settings\helloworld\Application Data\MathWorks\Matlab\
R2007a\compopts.bat
From template:              D:\Matlab~1\bin\win32\mbuildopts\msvc60compp.bat

Done . . .

--> "D:\Matlab~1\bin\win32\mwregsvr D:\Matlab~1\bin\win32\mwcomutil.dll"

DllRegisterServer in D:\Matlab~1\bin\win32\mwcomutil.dll succeeded
```

```
--> "D:\Matlab~1\bin\win32\mwregsvr D:\Matlab~1\bin\win32\mwcommgr.dll"

DllRegisterServer in D:\Matlab~1\bin\win32\mwcommgr.dll succeeded
```

2.4 mcc 编译器典型应用

2.4.1 独立可执行文件

Matlab 编译器最常见的应用就是将其编译为独立可执行文件，假设通过 Matlab 完成了一个图像显示程序，程序的代码如下：

```
function [] = img()
% function [] = img()
% 显示当前目录下文件名为 img.bmp 的文件
imshow('img.bmp');
```

在 Matlab 命令行输入如下命令：

```
mcc -m img.m
```

将 img.m 编译为独立可执行文件。编译完成后，如果查看一下当前目录，可以发现目录中多了几个文件(如表 2-1 所列)，这几个文件分别是：img.ctf、img.prj、img_main.c、img_mcc_component_data.c 和 img.exe，这些文件的功能和作用如表 2-1 所列。其中 img.ctf 和 img.exe 是发布独立可执行文件所必需的文件，img.prj、img_main.c 和 img_mcc_component_data.c 是编译 img.exe 时生成的中间文件。

表 2-1 img 工程编辑列表

文件名	功能和作用
img.ctf	ctf 文件，包含程序执行所需要的文件和数据
img.exe	编译的最终结果，是 img.m 文件的独立可执行文件
img.prj	工程文件
img_main.c	主函数接口文件
img_mcc_component_data.c	img_main.c 所需的相关数据

1. 编译过程

img.m 文件编译为独立可执行文件的过程如图 2-3 所示，整个编译过程的最终目标是生成 img.ctf 和 img.exe 文件，其中 img.exe 文件为最终独立可执行文件，img.ctf 文件则将 img.exe 所需的所有程序和数据打包放在了一起。

(1) img.ctf 文件的生成过程

① 编译命令传递给 Matlab 编译器。

② Matlab 编译器对 img.m 文件进行关联分析，目的是寻找执行 img.m 所调用的所有程序和数据。

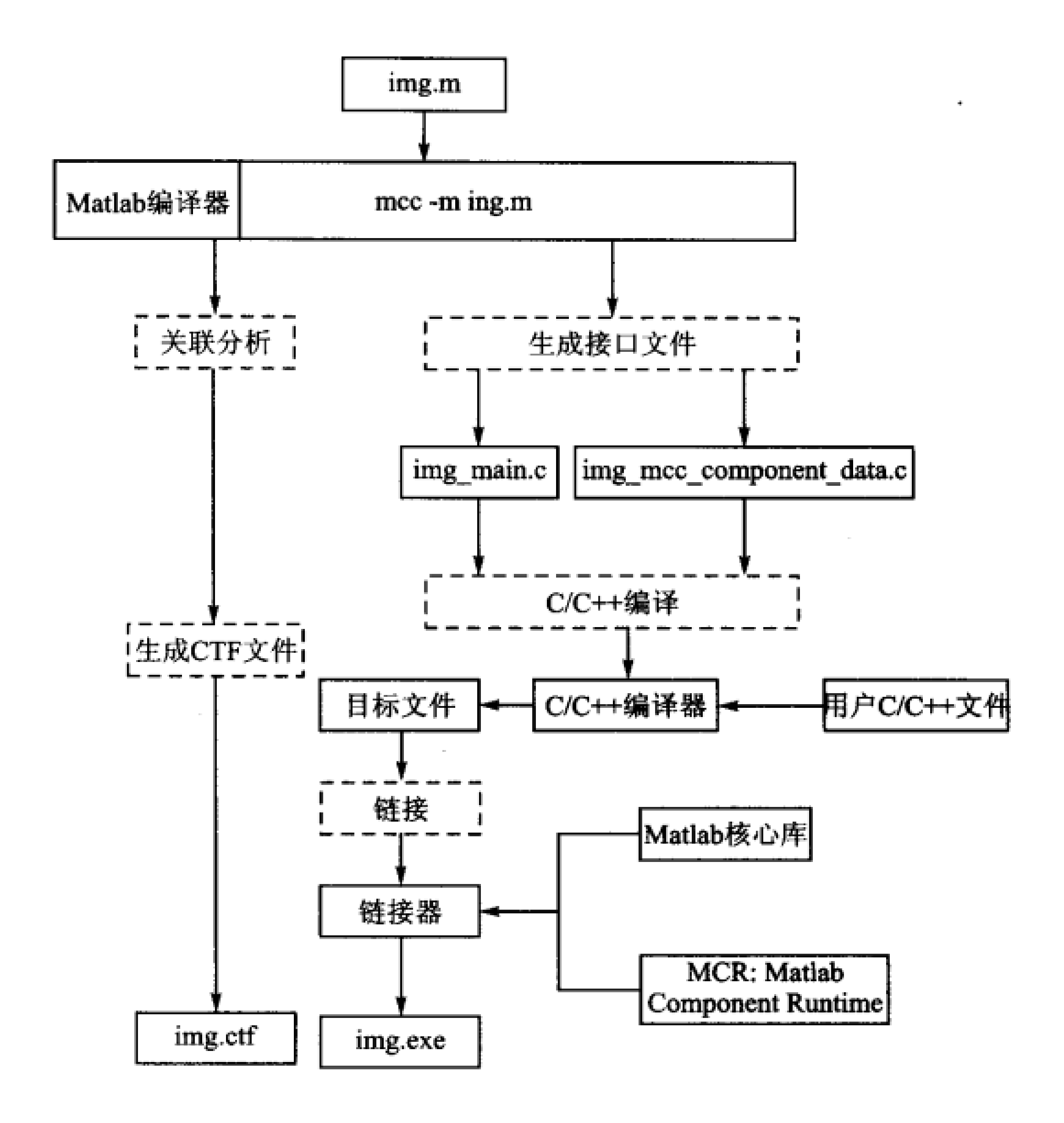


图 2-3 img.m 文件编译为独立可执行文件的过程

③ 根据关联分析的结果将 img.m 所需的所有程序和数据打包生成 img.ctf 文件。

(2) img.exe 文件的生成过程

① 编译命令传递给 Matlab 编译器。

② Matlab 编译器生成接口文件 img_main.c 和 img_mcc_component_data.c。

③ 这些接口文件连同用户自己编写的 C/C++文件(本例中没有用户自己编写的 C/C++文件)编译为目标文件。

④ 通过链接器将目标文件、Matlab 核心库和 MCR 链接生成最终的 img.exe 文件。

除了要生成接口文件以外,img.exe 的生成过程与普通的 C/C++程序的链接过程是一致的,因此从这里可以看出,Matlab 编译器只生成接口文件,而不是将 Matlab 程序全部编译为 C/C++文件。

到这里,读者可能会有新的疑问?到底 Matlab 编译器生成的接口文件是什么样的?下面就对 img.ctf、img_main.c、img_mcc_component_data.c 进行详细的剖析。

2. img.ctf 文件

由于 img.ctf 采用的是标准的压缩算法,因而为了了解 img.ctf,可以采用一种非常的手段,即将 img.ctf 用解压缩软件(例如 WinRar 软件)解压。解压以后的 img.ctf 包含如图 2-4 所示的内容,此时如果打开 img 文件夹,也许就能发现里面有编译过的 img.m 文件,并且 img.m 文件已经被加密,用 Matlab 代码编辑器打开以后会是一堆乱码。另外,打开 toolbox 目录,可以发现如

图 2-5 所示的内容,从名称上可以推测出这些文件是执行 img.m 文件时所需的 Matlab 工具箱中的文件。



图 2-4 img.ctf 解压后的文件目录内容

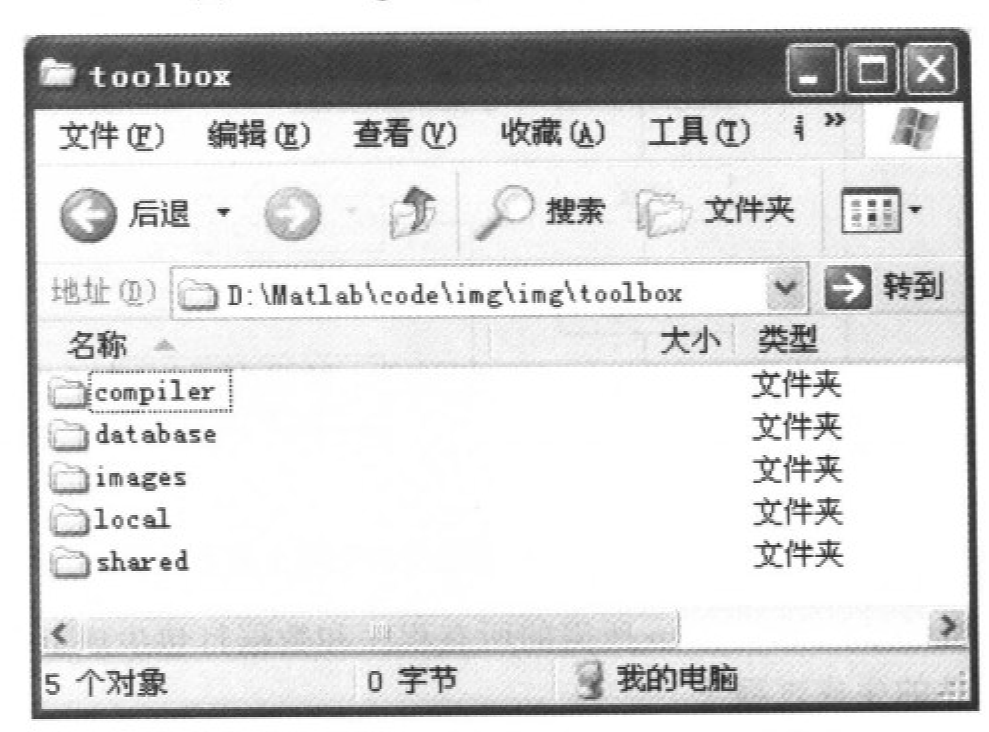


图 2-5 img.ctf 解压后 toolbox 目录中的内容

3. img_main.c

img_main.c 文件的代码以及其中包含的函数列表如下。

① static int mclDefaultPrintHandler(const char * s);

② static int mclDefaultErrorHandler(const char * s);

③ bool MW_CALL_CONV imgInitializeWithHandlers(
　　mclOutputHandlerFcn error_handler,
　　mclOutputHandlerFcn print_handler);

④ LIB_img_C_API bool MW_CALL_CONV imgInitialize(void);

⑤ LIB_img_C_API void MW_CALL_CONV imgTerminate(void);

⑥ int run_main(int argc,const char * * argv);

⑦ int main(int argc,const char * * argv);

这些函数的功能和调用顺序如图 2-6 所示。

从图 2-6 中可以看出,run_main 函数才是主体函数,其他函数则用来完成必要的程序初始化和程序终止任务。在 run_main 函数中,通过语句:

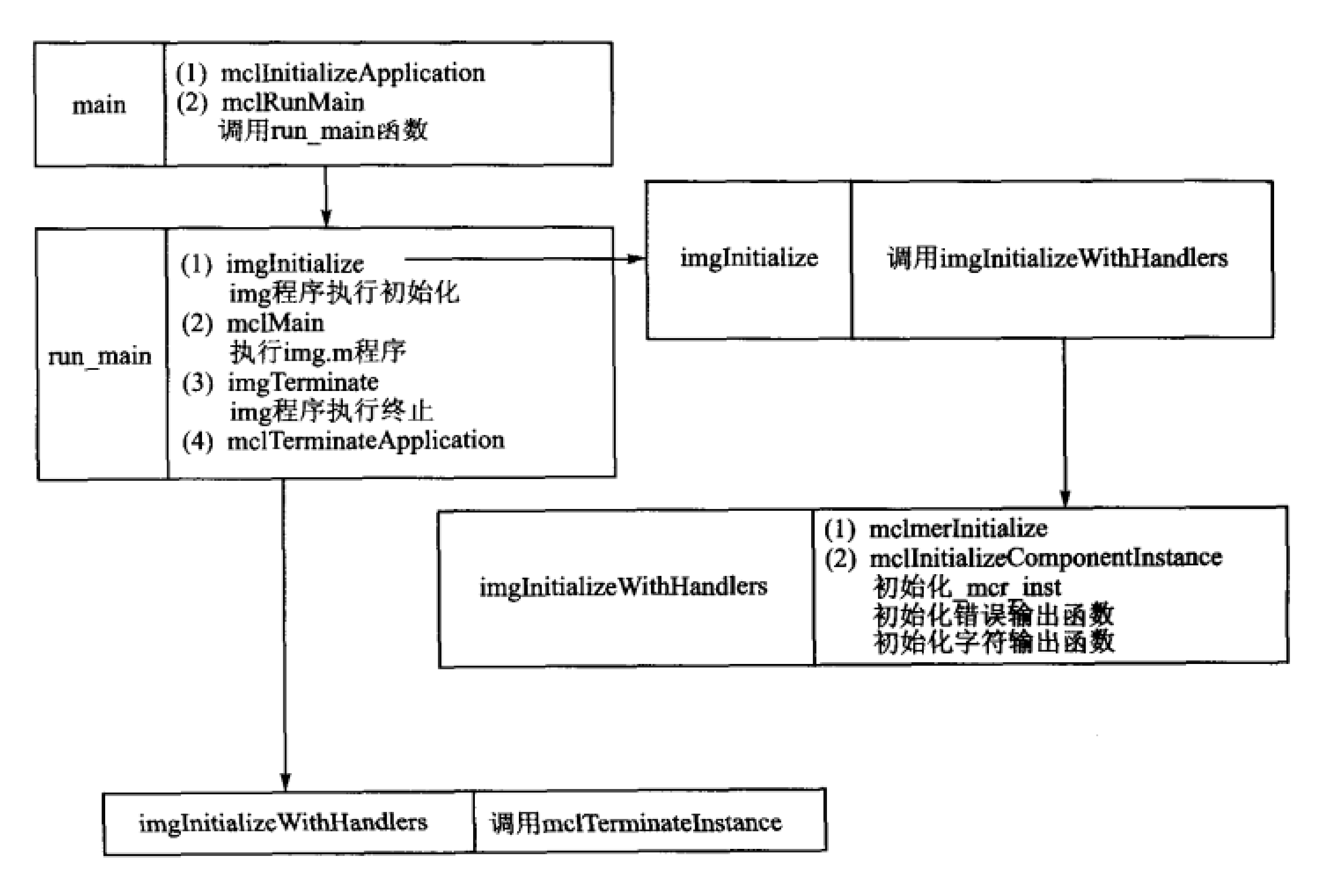


图 2-6　img_main.c 各主要函数功能和调用顺序

```
_retval = mclMain(_mcr_inst, argc, argv, "img", 0);
```

通知 MCR 执行 img.m 程序(img.m 存在于 img.ctf 文件中)。为了深入剖析这条语句的功能,在 extern\include\mclmcr.h 文件中,可以找到 mclMain 函数的定义如下:

```
EXTERN_C int mclMain(HMCRINSTANCE inst, int argc,
                                        const char * argv[], const char * name, int nlhs);
```

对于输入参数,其中:

```
HMCRINSTANCE inst;          //表示应用程序句柄
int argc;                   //应用程序命令个数
const char * argv[];        //应用程序命令
const char * name;          //通知 MCR 执行的 Matlab 程序名
int nlhs;                   //输出参数个数
```

可以看出,run_main 函数中对 mclMain 函数调用的功能可以解释为:在以_mcr_inst 句柄表示的应用程序(已经在 imgInitializeWithHandlers 中初始化)中通知 MCR 执行名称为 img.m 的 Matlab 程序,输出参数的个数为 0。

```
//img_main.c 代码
#include <stdio.h>
#include "mclmcr.h"
#ifdef __cplusplus
extern "C" {
#endif
```

```
extern mclComponentData __MCC_img_component_data;

#ifdef __cplusplus
}
#endif

static HMCRINSTANCE _mcr_inst = NULL;


#ifdef __cplusplus
extern "C" {
#endif

static int mclDefaultPrintHandler(const char *s)
{
    return mclWrite(1 /* stdout */, s, sizeof(char) * strlen(s));
}

#ifdef __cplusplus
} /* End extern "C" block */
#endif

#ifdef __cplusplus
extern "C" {
#endif

static int mclDefaultErrorHandler(const char *s)
{
    int written = 0;
    size_t len = 0;
    len = strlen(s);
    written = mclWrite(2 /* stderr */, s, sizeof(char) * len);
    if (len > 0 && s[ len - 1 ] != '\n')
        written += mclWrite(2 /* stderr */, "\n", sizeof(char));
    return written;
}

#ifdef __cplusplus
} /* End extern "C" block */
#endif

/* This symbol is defined in shared libraries. Define it here
 * (to nothing) in case this isn't a shared library.
 */
#ifndef LIB_img_C_API
#define LIB_img_C_API /* No special import/export declaration */
#endif
```

```
LIB_img_C_API
bool MW_CALL_CONV imgInitializeWithHandlers(
    mclOutputHandlerFcn error_handler,
    mclOutputHandlerFcn print_handler
)
{
    if (_mcr_inst != NULL)
        return true;
    if (!mclmcrInitialize())
        return false;
    if (!mclInitializeComponentInstance(&_mcr_inst,&__MCC_img_component_data,
                                        true,NoObjectType,ExeTarget,
                                        error_handler,print_handler))
        return false;
    return true;
}

LIB_img_C_API
bool MW_CALL_CONV imgInitialize(void)
{
    return imgInitializeWithHandlers(mclDefaultErrorHandler,
                                     mclDefaultPrintHandler);
}

LIB_img_C_API
void MW_CALL_CONV imgTerminate(void)
{
    if (_mcr_inst != NULL)
        mclTerminateInstance(&_mcr_inst);
}

int run_main(int argc,const char **argv)
{
    int _retval;
    /* Generate and populate the path_to_component. */
    char path_to_component[(PATH_MAX*2)+1];
    separatePathName(argv[0],path_to_component,(PATH_MAX*2)+1);
    __MCC_img_component_data.path_to_component = path_to_component;
    if (!imgInitialize()) {
        return -1;
    }
    _retval = mclMain(_mcr_inst,argc,argv,"img",0);
    if (_retval == 0 /* no error */) mclWaitForFiguresToDie(NULL);
    imgTerminate();
    mclTerminateApplication();
    return _retval;
```

```
}

int main(int argc,const char  * * argv)
{
    if (! mclInitializeApplication(
         __MCC_img_component_data.runtime_options,
         __MCC_img_component_data.runtime_option_count))
         return 0;

    return mclRunMain(run_main,argc,argv);
}
```

4. img_mcc_component_data.c

img_mcc_component_data.c 文件中,主要包含对结构体

```
mclComponentData __MCC_img_component_data
```

的初始化。__MCC_img_component_data 结构体包含了 img_main.c 程序所需要的所有初始化参数。当 Matlab 编译器编译 img.m 时,这些初始化参数以 const 常量的方式自动生成并放在 img_mcc_component_data.c 文件中,然后在声明__MCC_img_component_data 结构体变量时再用这些 const 常量初始化__MCC_img_component_data 结构体变量。关于__MCC_img_component_data 结构体变量各结构体域的含义,读者可以查看 extern\include\mclmcr.h 文件中关于 mclComponentData 结构体的定义说明。

2.4.2 C 动态链接库

采用 mcc -B csharedlib:img img.m 将 img.m 编译为动态链接库,编译完成以后,会生成如下文件:img.h、img.c、img.ctf、img.lib、img.dll、img.prj、img.exports、img.exp、img_mcc_component_data.c,其中 img.h 和 img.c 为动态链接库的接口文件,img.ctf 为生成的 ctf 文件,img.lib 和 img.dll 为动态链接库文件,img_mcc_component_data.c 为 MCR 初始化数据文件。这里,重点讲解 img.c 文件的组成,img.c 文件由下列函数组成:DllMain、mclDefaultPrintHandler、mclDefaultErrorHandler、imgInitializeWithHandlers、imgInitialize、imgTerminate、mlxImg、mlfImg,其中比较重要的函数如下所述。

1. DllMain

DllMain 是动态链接库的入口函数,在 Matlab 生成的 C++动态链接库中,DllMain 函数的主要作用是查找动态链接库所在的目录信息。MCR 可利用此目录信息查找动态链接库对应的 ctf 文件。

2. imgInitialize

imgInitialize 是初始化动态链接库。

3. imgTerminate

imgTerminate 是中止动态链接库。

4. mlxImg

mlxImg 与 mlfImg 函数的功能相同,即执行 img.m 文件的主要功能,但是其输入参数与 mexFunction 相同,其定义如下:

```
bool mlxImg(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[])
```

其中，

nlhs 表示输出参数个数，plhs 表示输出参数的 mxArray 数组；

nrhs 表示输入参数个数，prhs 表示输入参数的 mxArray 数组。

5. mlfImg

mlfImg 函数与 mlxImg 函数的功能相同，即执行 img. m 文件的主要功能，但其输入参数与 mlxImg 不同。img 函数的输入参数为逐个输入形式，而不是数组形式。本例中没有参数，假设 img. m 文件中 img 函数有一个输入参数 in 的话，实际生成的 img 函数定义如下：

```
mlfImg(mxArray * in)
```

img. h 包含 img. dll 动态链接库的导出函数声明，img. lib 包含 img. dll 动态链接库的符号表。在实际应用中，如果采用静态链接，需要 img. h、img. dll 和 img. lib 三个文件；如果采用动态链接，只需要 img. dll 一个文件。此时可以采用 Visual C++ 6.0 自带的 Depends 工具查看 img. dll 的导出函数名称列表，如图 2-7 所示，实际上 img. dll 导出了五个函数，即 mlfImg、imgInitialize、imgInitializeWithHandlers、imgTerminate 和 mlxImg。如果采用静态链接的话，则不需要考虑这些，而只要包含 img. h 头文件即可使用。但是，如果采用动态链接的话，则是要使用函数名得到 img. dll 导出函数的地址（参考附录中有关动态链接的相关知识）。

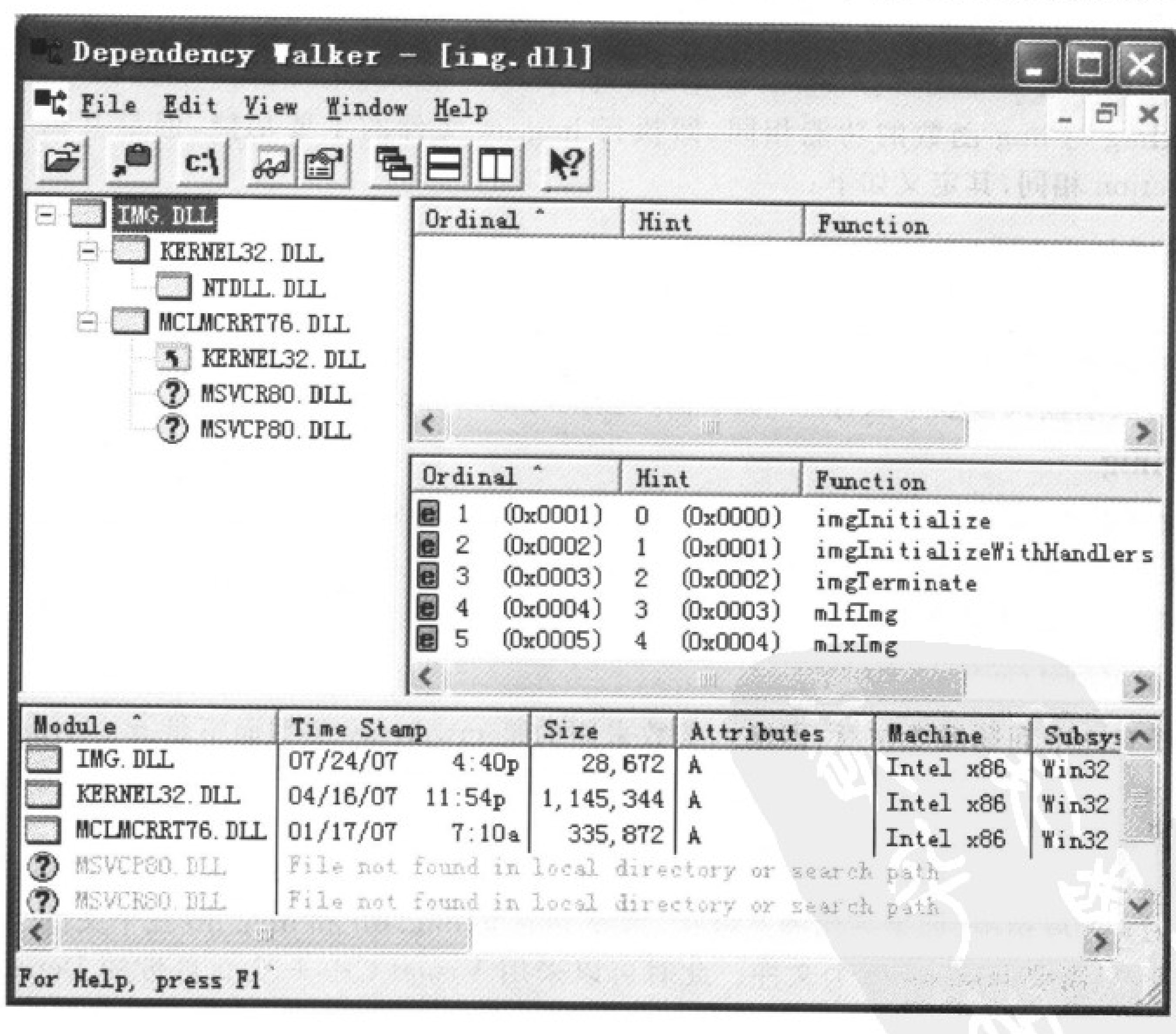


图 2-7　通过 Depends 工具分析 img. dll 的导出函数列表

2.4.3 C++动态链接库

采用 mcc -B cpplib:img img.m 将 img.m 编译为动态链接库，编译完成以后，会生成如下文件：img.h 、img.cpp、img.ctf、img.lib、img.dll、img.prj、img.exports、img.exp 和 img_mcc_component_data.c。其中 img.h 和 img.cpp 为动态链接库的接口文件，img.ctf 为生成的 ctf 文件，img.lib 和 img.dll 为动态链接库文件，img_mcc_component_data.c 为 MCR 初始化数据文件。这里，重点讲解 img.cpp 文件的组成。img.cpp 文件由下列函数组成：DllMain、mclDefaultPrintHandler、mclDefaultErrorHandler、imgInitializeWithHandlers、imgInitialize、imgTerminate、mlxImg 和 img，其中比较重要的函数如下所述。

1. DllMain

DllMain 是动态链接库的入口函数，在 Matlab 生成的 C++动态链接库中，DllMain 函数的主要作用是查找动态链接库所在的目录信息。MCR 利用此目录信息查找动态链接库对应的 ctf 文件。

2. imgInitialize

imgInitialize 是初始化动态链接库。

3. imgTerminate

imgTerminate 是中止动态链接库。

4. mlxImg

mlxImg 与 img 函数的功能相同，即执行 img.m 文件的主要功能，但是其输入参数与 mexFunction 相同，其定义如下：

```
bool mlxImg(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[])
```

其中，

nlhs 表示输出参数个数，plhs 表示输出参数的 mxArray 数组；

nrhs 表示输入参数个数，prhs 表示输入参数的 mxArray 数组。

5. img

img 函数与 mlxImg 函数的功能相同，即执行 img.m 文件的主要功能，但其输入参数与 mlxImg 不同。img 函数的输入参数为逐个输入形式，而不是数组形式。本例中没有参数，假设 img.m 文件中 img 函数有一个输入参数 in 的话，实际生成的 img 函数定义如下：

```
void img(const mwArray& in)
```

这里有一个问题需要注意，即输入参数采用的是 mwArray 类型而不是 mxArray 类型，这也是 C++动态链接库和 C 动态链接库的重要区别之一，关于 mwArray 类在 3.4 小节中会有详细讲解。

img.h 包含 img.dll 动态链接库的导出函数声明，img.lib 包含 img.dll 动态链接库的符号表。在实际应用中，如果采用静态链接，需要 img.h、img.dll 和 img.lib 三个文件；如果采用动态链接，只需要 img.dll 一个文件。此时可以采用 Visual C++ 6.0 自带的 Depends 工具查看 img.dll 的导出函数名称列表，如图 2-8 所示，实际上 img.dll 导出了五个函数，即?img@@YAXABVmwArray@@@Z、imgInitialize、imgInitializeWithHandlers、imgTerminate 和 mlxImg。如果采用静态链接的话，不需要考虑这些问题，只要包含 img.h 头文件即可使用。但是，如果采用动态链接的话，则是要考虑 img.dll 导出的函数中哪些是 C++标准的，而哪些

是 C 标准的。

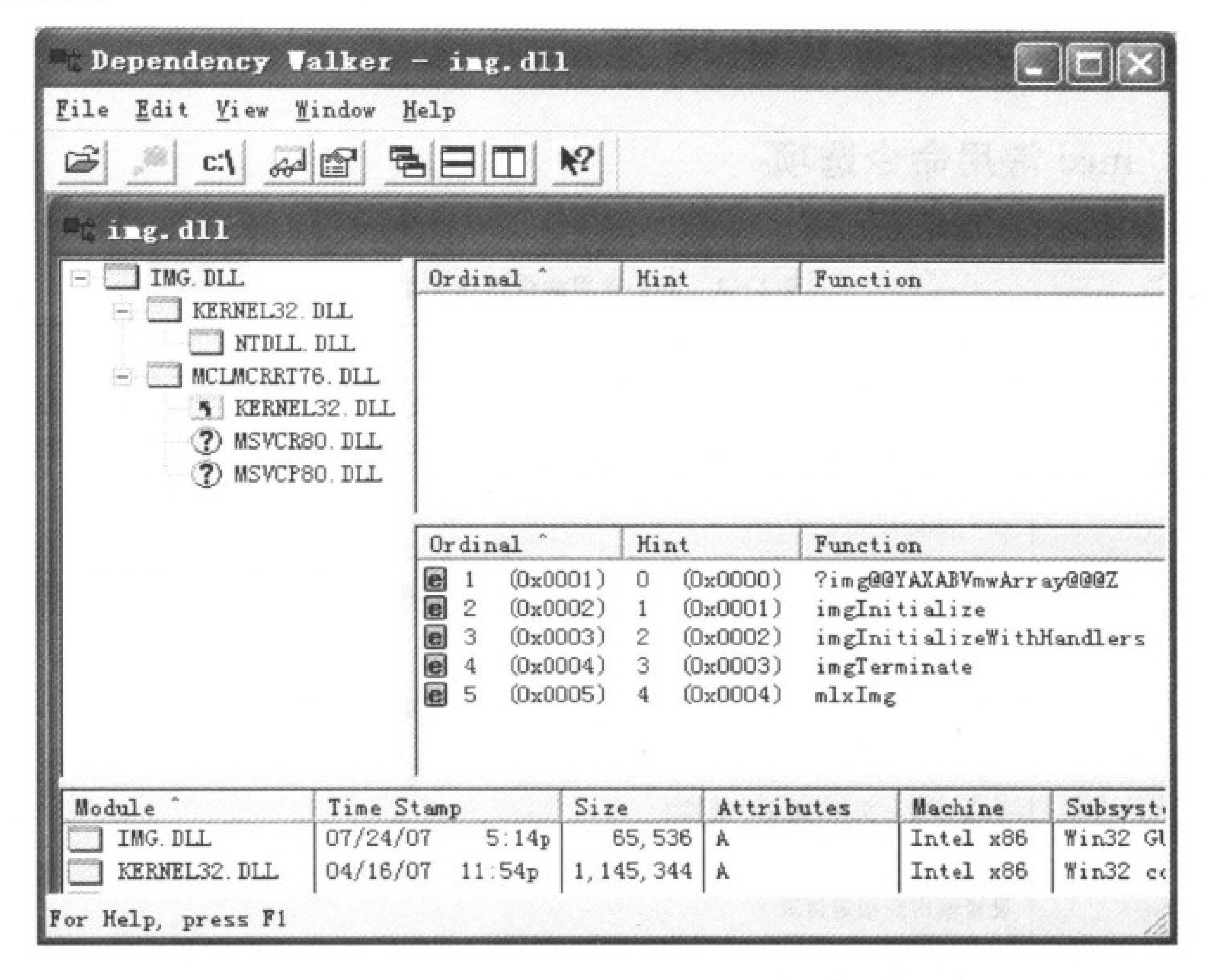


图 2-8　通过 Depends 工具分析 img.dll 的导出函数列表

2.4.4　C/C++动态链接库的不同之处

C/C++动态链接库之间是存在不同的，了解这些不同之处，才能决定到底哪种动态链接更符合实际应用程序的要求。C/C++动态链接库的不同之处总结如下所述。

1. 生成的接口文件语言不同

顾名思义，C 动态链接库的接口文件是 C 风格的；C++动态链接库的接口文件是 C++风格的。

2. 生成的动态链接库导出函数不同

C 动态链接库导出的主函数带有 mlf 前缀(比如 mlfImg)，而 C++动态链接库导出的主函数则不带 mlf 前缀(比如 img)。

3. 生成的动态链接库导出函数的风格不同

C 动态链接库导出函数都是 C 风格的；C++动态链接库主函数(比如，img 函数导出函数名为?img@@YAXABVmwArray@@@Z)导出采用 C++风格，其他函数采用 C 风格的。

4. 主函数输入输出的阵列类型不同

C 动态链接库中主函数输入输出的阵列类型都是 mxArray 类型，而 C++动态链接库中主函数输入输出的阵列类型都是 mwArray 类型。其中 mxArrray 是一个结构体类型，mwArray是一个 C++类类型，这两个阵列类型的操作方式是完全不同的，这一点请读者注意。

2.5 进一步了解 mcc 命令

2.5.1 mcc 常用命令选项

mcc 常用命令选项如表 2-2 所列。

表 2-2 mcc 常用命令选项列表

选 项	功能描述
-a filename	添加名称为 filename 的文件到 ctf 文件中，比如数据文件或者其他的程序文件，因为 Matlab 编译程序通过关联分析并不能发现程序执行所需的所有文件。比如要添加 data. mat 文件到 ctf 文件中，需要的语句为： mcc -a data. mat
-B	用以执行捆绑命令文件(Bundle File)，比如将 img. m 编译为名称为 img 的 c 动态链接库可以采用如下方式： mcc -B csharedlib:img img. m 将 img. m 编译为名称为 img 的 C++动态链接库则可以采用如下方式： mcc -B cpplib:img img. m
-c	生成 C 接口文件，此命令等同于： -T codegne
-d directory	设置输出到指定目录
-F project_name. prj	采用指定的工程文件进行编译，工程文件包含了所有命令参数，不需要再输入其他参数
-g	生成 debugging 信息
-I directory	采用 mcc 时增加 m 文件的搜索目录
-l	编译 C 动态链接库，此命令等同于： -W lib-T link:lib
-m	编译独立可执行文件，此命令等同于： -W main-T link:exe
-M string	向 mbuild 传递参数
-o outputifile	指定输出文件的名称
-R option	为 MCR 指定运行选项，其中 option 可以选择两个值： -nojvm -nojit
-v	编译时显示编译信息
-W type	控制生成接口文件的类型，其中 type 可以设定为：main，cpplib，lib，none，com 五种类型，最常用的为： main 独立可执行接口文件 cpplib C++动态链接库接口文件 lib C 动态链接库接口文件 none 不生成任何接口文件 com com 组件接口文件

续表 2－2

选　项	功能描述
-T target	控制编译器的编译输出类型，其中 target 可以设定为：codegen，compile：exe，compile：lib，link：exe，link：lib 五种类型，其中： codegen　表示生成 C/C＋＋接口文件 compile：exe　表示与 codegen 相同，并同时编译 compile：lib　表示与 codegen 相同，并同时编译 link：exe　表示与 complie：exe 相同，并同时链接为独立可执行文件 link：lib　表示与 compile：lib 相同，并同时链接为动态链接库文件
-z path	设置需要链接的库文件和头文件的路径

2.5.2　捆绑命令文件(bundle file)

Matlab 编译器命令 mcc 的参数比较多，为了简化命令输入方式，Matlab 编译器提供了捆绑命令文件(bundle file)机制，所有捆绑命令文件都在 toolbox\compiler\bundles 目录下。以 cpplib 为例，用文本编辑器打开 toolbox\compiler\bundles\cpplib 文件，文件的内容如下：

```
-W cpplib:%1% -T link:lib
```

其中，%1%表示输入参数。捆绑命令文件(bundle file)采用-B 选项调用，例如采用如下方式将 img.m 编译为名称为 img 的 C＋＋动态链接库：

```
mcc -B  cpplib:img img.m
```

命令"-B cpplib：img"的含义可以理解为用"-W cpplib：img-T link：lib"替代"-B　cpplib：img"，其中 img 为 cpplib 命令捆绑命令 cpplib 的输入参数。

在 toolbox\compiler\bundles 目录下，还存在很多捆绑命令文件，比如：

- macro_option_m：-W main-T link：exe
- csharedlib：-W lib：%1%-T link：lib

一般情况下，读者只需要知道如何使用这些捆绑命令文件即可，如果有必要，读者可以自行编写捆绑命令文件以降低使用 mcc 命令的复杂度。

2.6　Matlab 编译器高级应用

2.6.1　编译 script 文件

Matlab 编译器只能编译函数文件，如果确实需要编译 script 文件，可以把 script 文件转换为一个输入和输出参数都为空的函数文件。比如下列 script 文件代码：

```
%计算并显示 1 到 100 的和
v=1:100;
v_sum=sum(v(:));
disp(v_sum);
```

可以转换为如下的函数文件：

```
function [] = calsum100()
%计算并显示 1 到 100 的和
v=1:100;
v_sum=sum(v(:));
disp(v_sum);
```

当 script 文件转换为函数文件以后(假设函数文件的名字为 calsum100.m),可以调用 mcc 命令对其进行编译。

2.6.2 Matlab 编译器关联分析失效的情况

Matlab 编译器关联会分析会在某些情况下漏掉某些函数文件,这时候如果进行编译,无论是编译可执行文件还是编译动态链接库文件,最终都会导致运行失败。最常见的情况即编译图形界面 callback 函数、eval 函数和 feval 函数时会发生这种错误。下面分别以 callback 函数和 eval 函数为例说明。

1. callback 函数

以 callbacktest 函数为例,其代码如下所示。

```
% # function clickcallback
function [] = callbacktest()
% function [] = callbacktest()
% # function clickcallback
h=dialog;
pos=get(h,'position');
pos(3)=250;
pos(4)=250;
set(h,'position',pos);
hButton=uicontrol(h,'String','Click Me','CallBack','clickcallback','FontSize',35);
pos=get(hButton,'position');
pos(3)=200;
pos(4)=200;
set(hButton,'position',pos);

function clickcallback(hobj,eventdata)
msgbox('Button Clicked! ');
```

在 callbacktest 中建立了一个按钮控件,当按钮被单击时会调用名称为 clickcallback 的函数,clickcallback 函数存在于另一 M 文件 clickcallback.m 中。如果采用 mcc -m callbacktest.m 将 callbacktest.m 编译为独立可执行文件的话,编译过程不会有任何错误,但是执行时如果单击 Click Me 按钮(如图 2-9 所示)却会发现执行错误,错误信息为:

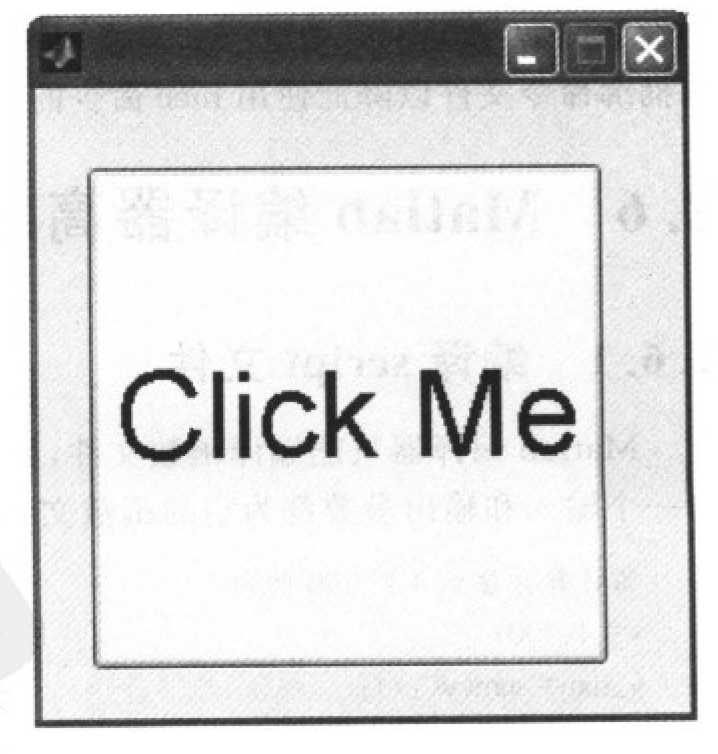


图 2-9　callbacktest 界面

```
??? Undefined function or variable 'clickcallback'
??? Error while evaluating uicontrol Callback
```

这个错误的发生表明 clickcallback.m 文件没有被打包到 callbacktest.ctf 文件中，即 Matlab 编译器没有检测到 clickcallback.m 与 callbacktest.m 的关联关系，此时 Matlab 编译器的关联分析失效了。

有三种办法可以解决这种问题。

第一种是采用 mcc 的-a 命令选项，把编译器没有检测到的 clickcallback.m 文件强制打包到 callbacktest.ctf 文件中。如果是编译为独立可执行文件，即采用如下命令行：

```
mcc -a clickcallback.m -m callbacktest.m
```

第二种是 callback 函数采用函数句柄的方式调用，即语句：

```
hButton = uicontrol(h,'String','Click Me','CallBack','clickcallback','FontSize',35);
```

可以更改为(请注意带下划线的语句)：

```
hButton = uicontrol(h,'String','Click Me','CallBack',@clickcallback,'FontSize',35);
```

第三种是在主函数 callbacktest.m 中通过%#function 标志显式通知编译器哪些函数需要编译并打包到 ctf 文件中。对于本例，只需在 callbacktest.m 文件函数实体的开始加上该标志语句然后再编译即可。

2. eval、feval 函数

编译带有 eval 和 feval 函数的 Matlab 程序会出现与 callback 函数同样问题。以 eval 函数为例，在 evaltest.m 文件中，通过 eval 函数调用了 showdlg 函数。如果采用 mcc -m evaltest.m 将 evaltest.m 编译为独立可执行文件的话，编译过程不会有任何错误，但是执行时会发生如下错误：

```
??? Undefined function or variable  'clickcallback'
??? Error while evaluating uicontrol callback
```

引起这种问题的原因同样是由于 Matlab 编译器的关联分析失效，showdlg.m 文件没有被打包到 evaltest.ctf 文件中，解决方法与 callback 函数的解决方法类似，读者可以以 evaltest.m 为例练习一下。

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%evaltest.m 文件
function [] = evaltest()
% function [] = evaltest()
eval('showdlg');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%showdlg.m 文件
function [] = showdlg;
h = dialog;
```

```
pos = get(h,'position');
pos(3) = 250;
pos(4) = 250;
set(h,'position',pos);
hButton = uicontrol(h,'String','Click Me','CallBack',@click,'FontSize',35);
pos = get(hButton,'position');
pos(3) = 200;
pos(4) = 200;
set(hButton,'position',pos);

function click(hobj,eventdata)
msgbox('Button Clicked! ');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.6.3 从 C/C++中调用 Matlab 内置函数(built-in function)

Matlab 的内置函数(built-in function)没有给出 C/C++接口,但是可以通过 M 文件调用 Matlab 内置函数。例如,如果用户需要调用 magic 内置函数,则可以编写下面的 magicsquare.m 文件间接调用 magic 内置函数。magicsquare.m 文件完成的功能就是嵌套了 magic 内置函数的调用。通过 mcc -csharedlib:magicsquare magicsquare.m 将其编译为 C 动态链接库或者通过 mcc -cpplib:magicsquare magicsquare.m 将其编译为C++动态链接库,然后再通过 C/C++程序调用动态链接库即可完成 magic 内置函数的调用。

```
function m = magicsquare(n)
% function m = magicsquare(n)
% 调用 Matlab 内置函数 magic 生成魔方数组
% 魔方数组的行列和对角线元素的和相同
if ischar(n)
    n = str2num(n);
end
m = magic(n);
```

2.6.4 可变参数传递(varargin,varargout)

Matlab 语言的一个重要的特点就是可以输入和输出可变参数,其中输入可变参数采用 varargin 元组阵列表示,而输出参数采用 varargout 元组阵列表示。但是 C/C++函数的输入和输出参数则必须是固定,那么如果采用可变参数的 Matlab 程序编译为动态链接库以后,如何通过 C/C++来调用呢?下面以 var_uncertain 函数为例,来说明这个问题。

var_uncertain 函数的输入为 x,y 加上一个可变参数 varargin,输出为 a,b 加上一个可变参数,函数实体不实现任何功能,只是将输入参数原样传递给输出参数。采用 mcc -B csharedlib:var_uncertain var_uncertain.m 将其编译为 C 动态链接库以后,打开 var_uncertain.h 文件,可以看出 var_uncertain 函数的接口函数为:

```
bool mlfVar_uncertain(
                    int nargout,mxArray * * a,mxArray * * b,
                    mxArray * * varargout,mxArray * x,
                    mxArray * y,mxArray * varargin)
```

通过观察,可以看出对于输出参数通过 nargout 确定总的输出参数个数,其中 mxArray * * varargout 用来表示可变的输出参数,其个数用 nargout－2 来确定(减 2 表示输出参数中确定的参数为两个)。而输入参数的个数没有在程序明确表示,因而在使用时需要特别注意,尤其在初始化 mxArray * varargin 元组阵列时更要注意。因而,一般情况下不建议使用可变输入参数,因为稍有不慎就会导致程序产生致命的错误。如果确实要使用可变输入参数,程序开发人员要在 Matlab 程序对各种情况进行出错处理,而且在发布 Matlab 编译器编译后的动态链接库时,需要详细说明输入参数的类型、个数以及初始化的方法等。

如果采用 mcc -B cpplib:var_uncertain_cpp var_uncertain.m 将其编译为 C++动态链接库的话,生成的 var_uncertain 接口函数则略有不同,如下所示:

```
var_uncertain(int nargout,mwArray& a,mwArray& b,
            mwArray& varargout,const mwArray& x,
            const mwArray& y,const mwArray& varargin);
```

其处理可变输入输出参数的方法与 C 接口函数 mlfVar_uncertain 是一致的,只是采用的是 mwArray 类而不是 mxArray 接口体表示 Matlab 阵列。

```
function [a,b,varargout] = var_uncertain(x,y,varargin)
% function [a,b,varargout] = var_uncertain(x,y,varargin)
a = x;
b = y;
no = length(varargout(:));
ni = length(varargin(:));

n = min(ni,no);
for i = 1:n
    varargout(i) = varargin(i);
end

if no>ni
   for i = (n+1):no
    varargout(i) = [];
   end
else
end
```

2.6.5 Matlab 环境下执行和 MCR 执行的不同之处

在 Matlab 执行 m 程序与经过 Matlab 编译器编译后通过 MCR 执行 m 程序基本相同,但

是有些功能在 Matlab 环境下可以执行，但是通过 MCR 执行却会发生错误。比如 addpath 函数和 print 函数，这一点需要引起读者注意。如果程序调用了这些函数，就需要做一些特殊处理，否则编译后通过 MCR 执行的时候就会报错。幸好 Matlab 提供了解决这种问题的方法，使用 isdeployed 函数可以判断当前 M 文件到底是在 Matlab 还是 MCR 环境下执行，然后特殊情况特殊处理。以 print 函数(打印 figure 窗口)为例，Matlab 提供了 deploypint 函数作为替代。如下的 print_figure 函数就提供了一个分别在 Matlab 环境和 MCR 环境处理的实例。程序采用 isdeployed 判断 m 程序到底在 Matlab 还是 MCR 环境执行，如果在 MCR 环境下执行采用 deployprint 函数替代 print 函数。

```
function [] = print_figure(hf)
% function [] = print_figure(hf)
if nargin == 0
    hf = figure;
    sphere;
else
end

if isdeployed %MCR 运行
    deployprint(hf);
else %Matlab 运行
    print(hf);
end
isdeployed
deployprint
```

2.6.6 获取 CTF 文件的目录

编写 Matlab 程序的时候，有时候需要获取编译后 Matlab 程序通过 MCR 运行时对应的 CTF 文件目录。Matlab 提供了 ctfroot 函数用以获取 CTF 文件的目录。如果在程序中需要获取 CTF 文件的目录，只需要调用 ctfroot 函数即可。如下所述，在 showctfroot. m 文件中，实现的功能即获取 CTF 文件的目录并显示出来。

如果 showctfroot. m 编译为 C++动态链接库，用户可以通过 showctfroot 对应的接口函数获取 CTF 文件的路径信息，其中 showctfroot 对应的接口函数为：

```
bool mlxShowctfroot(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[]);
void showctfroot(int nargout,mwArray& ctfpath);
```

如果 showctfroot. m 编译为 C 动态链接库，可以采用类似操作。

```
function [] = testctfroot()
showctfroot;

function [ctfpath] = showctfroot()
% function [ctfpath] = showctfroot()
ctfpath = ctfroot;
```

```
disp(ctfpath);
```

2.6.7 屏幕打印和错误信息显示函数

编译 Matlab 程序时，会在主程序接口文件中看到 mclDefaultPrintHandler 和 mclDefaultErrorHandler 这两个函数，它们在 print_error_handle_testInitialize 函数中初始化为程序默认的屏幕打印和错误信息显示函数。如果用户需要，可以自行定制这两个函数，即自己编写或者改动这两个函数（其实接口文件中的所有函数都可以进行改动或者重新编写，然后再利用 mbuild 编译器编译）。下面以 print_error_handle_test.m 为例，说明屏幕打印和错误信息显示函数的处理方法。

```
function [] = print_error_handle_test()
% function [] = print_error_handle_test()
a = magic(3);
disp(a);
error('print_error_handle_test 函数在此发生错误');
```

在 print_error_handle_test 函数中，笔者刻意安排了屏幕打印和错误信息显示这两种情况，其中 disp 函数负责输出阵列 a 的信息，error 函数负责输出错误信息。首先，采用 mcc -m print_error_handle_test.m 命令将其编译为独立可执行文件。这时候如果打开 print_error_handle_test_main.c 文件的话，可以看到 mclDefaultPrintHandler、mclDefaultErrorHandler、print_error_handle_testInitialize 三个函数，默认的情况下所有的屏幕打印和错误信息都输出在标准设备即显示器屏幕上。假如希望所有的信息能够输出到文件中，以方便日后查看，就可以更改这三个函数。为了实现将屏幕打印和错误信息都输出到文件中（output.txt），可对 print_error_handle_test_main.c 进行了改动，改动如下所述。

① 增加一个 static 全局变量，用来存储文件指针。

```
static FILE * fp = NULL;
```

② 重新编写屏幕信息输出函数，并命名为 myDefaultPrintHandler。

```
static int myDefaultPrintHandler(const char * s)
```

③ 在 mclDefaultErrorHandler 中增加输出到文件的语句

```
fprintf(fp, "%s", s);
```

④ 在 main 函数中增加对文件指针的初始化和销毁代码

```
if(fp == NULL)
    {
        fp = fopen("output.txt", "a");
    }
mclRunMain(run_main, argc, argv);
if(fp != NULL)
    {
        fclose(fp);
```

```
        }
```

改动后的 print_error_handle_test_main.c 代码如下所示，其中与自动生成的代码不同或者增加之处用灰色背景加深，便于读者查找。

```
/*
 * Matlab Compiler: 4.6 (R2007a)
 * Date: Wed Jul 25 12:33:27 2007
 * Arguments: "-B" "macro_default" "-m" "-W" "main" "-T" "link:exe"
 * "print_error_handle_test.m"
 */

#include <stdio.h>
#include "mclmcr.h"
#ifdef __cplusplus
extern "C" {
#endif

extern mclComponentData __MCC_print_error_handle_test_component_data;

#ifdef __cplusplus
}
#endif

static HMCRINSTANCE _mcr_inst = NULL;

#ifdef __cplusplus
extern "C" {
#endif

static int mclDefaultPrintHandler(const char * s)
{
    return mclWrite(1 /* stdout */,s,sizeof(char) * strlen(s));
}

static FILE * fp = NULL;
static int myDefaultPrintHandler(const char * s)
{
    fprintf(fp,"%s",s);
    return 1;
}

#ifdef __cplusplus
} /* End extern "C" block */
#endif

#ifdef __cplusplus
extern "C" {
```

```
#endif

static int mclDefaultErrorHandler(const char * s)
{
    int written = 0;
    size_t len = 0;
    len = strlen(s);
    fprintf(fp,"%s",s);
    written = mclWrite(2 /* stderr */,s,sizeof(char) * len);
    if (len > 0 && s[ len - 1 ] != '\n')
        written += mclWrite(2 /* stderr */,"\n",sizeof(char));
    return written;
}

#ifdef __cplusplus
} /* End extern "C" block */
#endif

/* This symbol is defined in shared libraries. Define it here
 * (to nothing) in case this isn't a shared library.
 */
#ifndef LIB_print_error_handle_test_C_API
#define LIB_print_error_handle_test_C_API /* No special import/export declaration */
#endif

LIB_print_error_handle_test_C_API
bool MW_CALL_CONV print_error_handle_testInitializeWithHandlers(
    mclOutputHandlerFcn error_handler,
    mclOutputHandlerFcn print_handler
)
{
    if (_mcr_inst != NULL)
        return true;
    if (!mclmcrInitialize())
        return false;
    if (!mclInitializeComponentInstance(&_mcr_inst,
                                        &__MCC_print_error_handle_test_component_data,
                                        true,NoObjectType,ExeTarget,
                                        error_handler,print_handler))
        return false;
    return true;
}

/*LIB_print_error_handle_test_C_API
bool MW_CALL_CONV print_error_handle_testInitialize(void)
{
    return
```

```
print_error_handle_testInitializeWithHandlers(mclDefaultErrorHandler,
    mclDefaultPrintHandler);
} */

LIB_print_error_handle_test_C_API
bool MW_CALL_CONV print_error_handle_testInitialize(void)
{
    return
    print_error_handle_testInitializeWithHandlers(mclDefaultErrorHandler,
                                                  myDefaultPrintHandler);
}

LIB_print_error_handle_test_C_API
void MW_CALL_CONV print_error_handle_testTerminate(void)
{
    if (_mcr_inst != NULL)
        mclTerminateInstance(&_mcr_inst);
}

int run_main(int argc,const char * * argv)
{
    int _retval;
    /* Generate and populate the path_to_component. */
    char path_to_component[(PATH_MAX * 2) + 1];
    separatePathName(argv[0],path_to_component,(PATH_MAX * 2) + 1);
    __MCC_print_error_handle_test_component_data.path_to_component = path_to_component;
    if (! print_error_handle_testInitialize()) {
        return -1;
    }
    _retval = mclMain(_mcr_inst,argc,argv,"print_error_handle_test",0);
    if (_retval == 0 /* no error */) mclWaitForFiguresToDie(NULL);
    print_error_handle_testTerminate();
    mclTerminateApplication();
    return _retval;
}

int main(int argc,const char * * argv)
{
    if (! mclInitializeApplication(
        __MCC_print_error_handle_test_component_data.runtime_options,
        __MCC_print_error_handle_test_component_data.runtime_option_count))
        return 0;

    if(fp == NULL)
        {
            fp = fopen("output.txt","a");
```

```
    }

    mclRunMain(run_main,argc,argv);

    if(fp! =NULL)
        {
            fclose(fp);
        }

    return 0;
}
```

2.7　Deployment Tool

在 Matlab 命令行中输入 deploytool 可以启动 Deployment Tool 工具，如图 2－10 所示。Deployment Tool 实际上是 Matlab 编译器的一个集成 GUI 界面，通过 Deployment Tool 可避免用户输入繁琐的 Matlab 编译器指令，从而提高工作效率，降低使用 Matlab 编译器的难度。

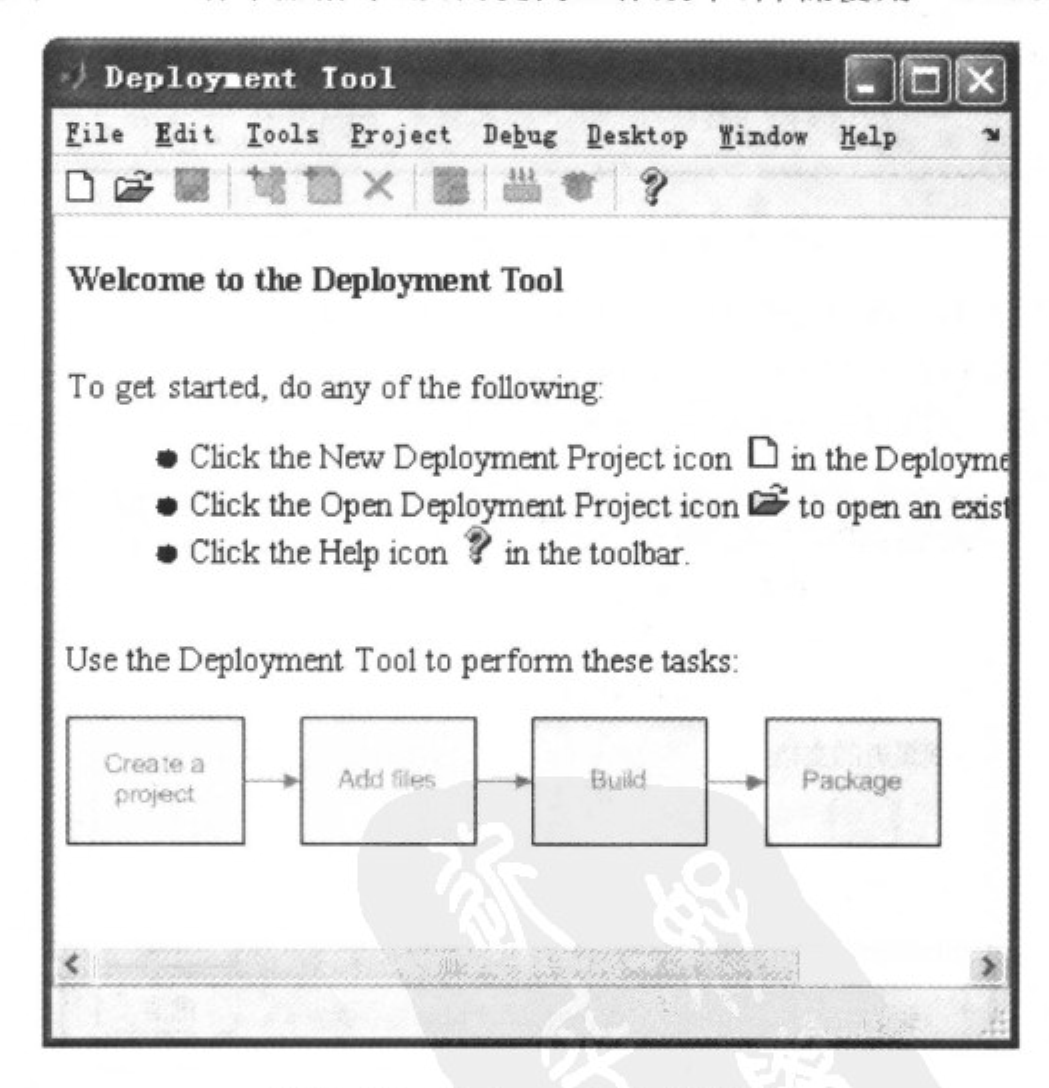


图 2－10　Deployment Tool 对话框

选择 File|New Deployment Project 菜单项，弹出如图 2－11 所示的对话框，从图中可以看出 Deployment Tool 可以完成 Matlab 编译器、Matlab Excel Builder 和 Dotnet Builder 等的功能。Mathworks 在 Matlab 2007 的文档中也提到，未来 Deployment Tool 将取代 Matlab 编译器、Matlab Excel Builder、Dotnet Builder 现有的图形界面工具，比如 dotnetool 在未来版本会被 Deploy-

ment Tool 所取代(实际上使用方法类似,只是将操作界面集中于 Deploment Tool)。

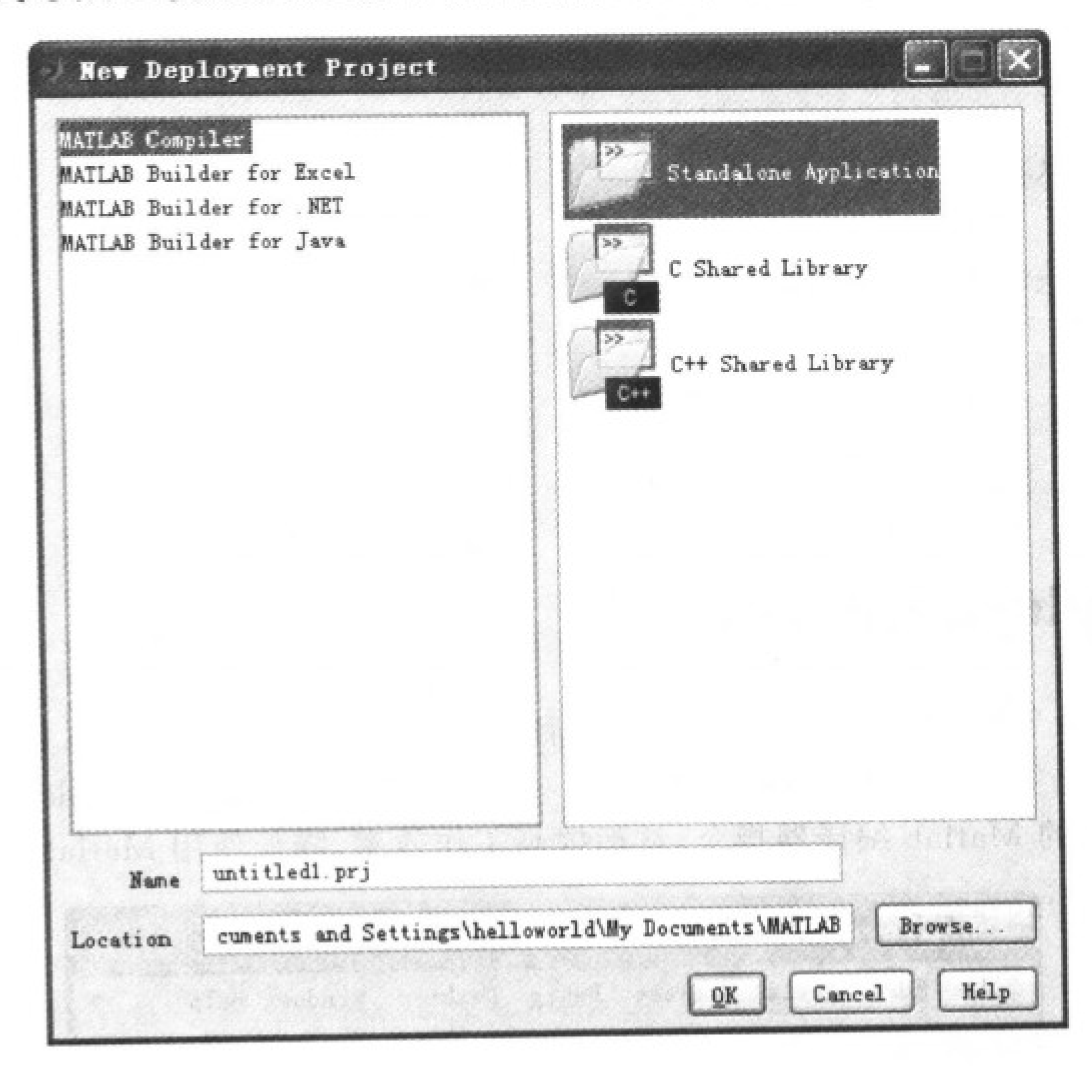


图 2-11　新建 Deployment Tool 工程

以独立可执行文件为例,说明 Deployment Tool 的使用方法。

① 建立一个 Standalone Application 新工程。

② 向工程中添加需要编译的文件,如图 2-12 所示。



图 2-12　为工程添加文件

③ 选择 Tools|Build 菜单项编译独立可执行文件工程。

④ 选择 Tools|Package 菜单项将生成的可执行文件和 ctf 文件打包为〈projetname〉.exe 文件(自解压可执行文件)。用户只需要在目标机器上运行〈projetname〉.exe 即可将新生成的独立可执行文件安装到目标机器上。

通过 Deployment Tool 建立其他类型的工程如 C 动态链接库、C＋＋动态链接库及 Dotnet Builder等的步骤与独立可执行文件工程类似，读者可以参考上述步骤进行。

2.8　程序发布

如果要发布 Matlab 编译器生成的文件，无论是独立可执行文件还是动态链接库文件，用户都要在目标机上安装 MCR。MCR 安装通过 MCRInstaller. exe 来实现，其中 MCRInstaller. exe 可以通过 buildmcr 命令来生成。默认情况下，用户可以在〈matlabroot〉\toolbox\compiler\deploy\win32 目录下找到 MCRInstaller. exe 文件，一般情况下此 MCRInstaller. exe 文件都能满足用户的需求。用户只需要在目标机器上运行 MCRInstaller. exe 即可安装 MCR。

除了处理安装 MCR 之外，用户还需要将 Matlab 编译器生成的目标文件复制或安装到目标机器上。对于可执行文件而言，目标文件即. ctf 文件和. exe 文件；对于动态链接库文件而言，目标文件即. ctf 文件、. dll 文件、. lib 文件和. h 文件。如果使用 Deployment Tool 的话，通过 Tool|Package 菜单项可以将生成的所有目标文件打包为一个自解压的 exe 文件，方便用户使用。



第 3 章　Matlab 与 C 语言的接口

3.1　Matlab C/C++编译器的设置(mex)

Matlab 可以将 C 程序编译为 MEX 文件供 Matlab 直接调用(在 Windows 平台下,MEX 文件实质上是动态链接库,Matlab 6.5 以前的版本采用 dll 为扩展名;Maltab 7.0 及以后采用 mexw32 或 mexw64 为扩展名)。Matlab C/C++编译器的设置包括 mbuild -setup 和 mex -setup两个步骤,在 2.3 节中介绍了 mbuild -setup 的配置过程,这里介绍 mex -setup 的配置过程。

在 Matlab 的 Command Window 下输入命令 mex -setup,并根据 Matlab 的提示选择合适的选项,如下所示(其中加粗部分为用户输入部分)。

```
>>mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in D:\Matlab~1\sys\lcc
[2] Microsoft Visual C++ 6.0 in D:\Program Files\Microsoft Visual Studio

[0] None

Compiler: 2

Please verify your choices:

Compiler: Microsoft Visual C++ 6.0
Location: D:\Program Files\Microsoft Visual Studio

Are these correct? ([y]/n): y

Trying to update options file: C:\Documents and Settings\helloworld\Application Data\MathWorks\Matlab\R2007a\mexopts.bat
From template:              D:\Matlab~1\bin\win32\mexopts\msvc60opts.bat

Done . . .
```

3.2　Matlab 中调用 C 程序-MEX 文件

3.2.1　MEX 文件介绍

1. 目　的

在 Matlab 开发环境中调用 C/C++等外部程序需要借助编译器将 C/C++代码编译为 MEX 文件以后才能实现，其中 MEX 文件包含 Matlab 解释器可以动态装载和执行动态链接模块。在 Windows 平台下，MEX 文件以动态链接库(根据 Matlab 版本的不同，扩展名为 *.dll、*.mexw32 和 *.mexw64)的形式存在。通过用 C 语言编写代码，然后通过 Matlab 编译器将其编译为 MEX 文件可以达到下面一些目的：

(1) 加快程序的执行速度

采用 Matlab 开发一些仿真程序时，如果其中有的模块在 Matlab 中的执行效率很低(如循环)，可以采用 MEX 文件的方式用 C 语言来实现这些效率比较低的模块，从而提高整个程序的执行速度。

(2) 将 Matlab 作为 C 语言开发的调试环境

在 Matlab 下面进行数据显示是非常方便的，因而可以将一些在其他开发环境中不方便数据显示的函数用 C 语言写好以后，通过 MEX 文件方式在 Matlab 环境下进行调试。尤其是有大量数据需要处理时，用 Matlab 观察其中间结果是非常方便的。

(3) 扩展 Matlab 的功能

Matlab 具有强大的矩阵运算能力并且拥有丰富的工具箱，但是在有些方面比较薄弱，比如硬件设备接口操作和图形化程序设计等，用户可以通过 MEX 文件利用 C/C++语言扩展 Matlab 的薄弱环节，以满足自己的需求。

2. 注意事项

只要按照 Matlab 规定的书写格式编写 C/C++程序，那么生成的 MEX 文件的调用方法与一般的 Matlab 函数完全一样。在使用 MEX 文件时，需要注意下面几点：

① MEX 文件的执行优先级比一般的 Matlab 函数要高，因而如果一个目录下有 Matlab 函数的名称与 MEX 文件相同，则 MEX 文件会被调用，而相应的 Matlab 函数由于执行优先级较低则不会被调用。

② 如果想让 MEX 文件像一般的 Matlab 函数那样，用 help 命令可以看到这个函数的帮助信息，可以采用在与 MEX 文件相同目录下放置名称相同的 M 文件，并将帮助信息放在 M 文件中的办法来解决。因为 help 命令只能显示 M 文件的帮助信息。

3. 编译示例

为了便于读者理解，下面举例说明 MEX 文件的编译过程。

(1) 编写 C 语言文件

```
/* helloworld.c */
#include "mex.h"
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,
                 const mxArray *prhs[])
{
```

```
    mexPrintf("Hello World! \n");
}
```

(2) 将 C 语言文件编译为 MEX 文件

在 Matlab 命令行下,执行命令 mex filename. c 将编写的 C 语言文件编译为 MEX 文件。如果这个例子的 C 语言文件名为 helloworld. c,则用 mex helloworld. c 将其编译为 MEX 文件。编译成功后,可以在与 helloworld. c 文件相同的目录下找到 helloworld. mexw32 文件,这便是编译成功的 MEX 文件。

(3) 执行可执行文件

在 Matlab 命令行中执行 helloworld,就会在屏幕上打印出"Hello World!"字样。

3.2.2 MEX 文件结构说明

用 C 语言编写 MEX 文件的源代码时,必须要有 mexFunction 函数。mexFunction 函数的作用与一般 C 语言程序设计中的 main 函数的功能类似。如果说 main 函数提供的是操作系统与 C 语言子程序之间的接口,那么 mexFunction 函数的作用则是 Matlab 与 C 语言子程序之间的接口。另外,用 C 语言编写 MEX 文件源代码时,需要包含"mex. h"头文件,即

```
#include "mex.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[]);
```

其中,mexFunction 函数的输入/输出参数的含义如表 3-1 所列。

表 3-1 mexFunction 输入参数

参数名	含 义	参数名	含 义
int nlhs	输出参数的个数	int nrhs	输入参数的个数
mxArray * plhs	输出参数的 mxArray 数组	mxArray * prhs	输入参数的 mxArray 数组

在 Matlab 中,所有的数据类型都使用 mxArray 结构来表示。通过接口函数 mexFunction,可以与 Matlab 环境进行数据交换。Matlab 与 mexFunction 数据交互的过程可以用图 3-1 来说明,从图中可以看出,输入参数用 nrhs 和 prhs 两个量来描述。prhs 是一个 mxArray 结构的指针

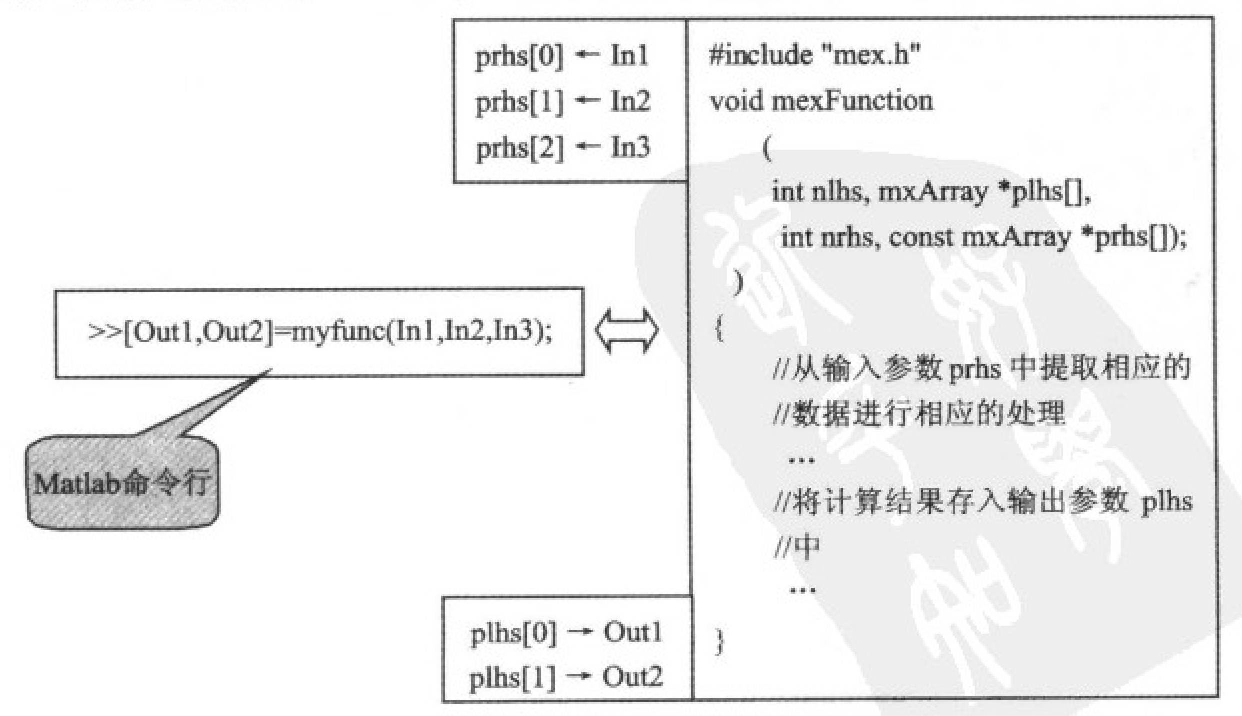


图 3-1 mexFunction 与 Matlab 的数据交换示意图

数组,而 nrhs 表示这个数组的大小,即输入参数的个数。同样,输出参数用 plhs 和 nlhs 来描述。其中 plhs 是一个 mxArray 结构的指针数组,而 nlhs 则表示 plhs 的大小,即输出参数的个数。

3.3　编译 MEX 文件

在 Matlab 命令行中通过 mex 命令编译 MEX 文件,假设要编译 filename. c 文件,只需要在命令行中输入 mex filename. c 命令即可。mex 命令有很多选项,如果用户有特殊的应用,可以在调用 mex 命令时设置这些选项,比如采用-largeArrayDims 选项意味着 MEX 文件中可以使用长度大于 2^31－1 的阵列,其他更多的 mex 命令选项用户可以查看 Matlab 帮助(一般情况采用默认选项即可)。

3.4　Matlab 中 mxArray 类型的操作

从图 3-1 中可以看出,用 C 语言编写 MEX 文件的一个关键之处在于 mexFunction 函数中关于 Matlab 与 C 代码模块的数据交互问题。Matlab 所有的数据类型都可以用 mxArray 来描述,并且 mexFunction 函数的所有输入输出参数都是采用 mxArray 形式来实现的。那么如何将 mxArray 类型转换为 C 语言可以直接使用的基本数据类型(如 double,int 等),或者如何将 C 语言的基本数据类型转换为 mxArray 类型?这个问题 Matlab 通过提供一系列操作 mxArray 的 API 函数得以解决。Matlab 与 C 混合编程时需要调用 Matlab 提供的一些 API 函数,其中以 mx 开头的 Matlab API 函数主要是提供对 mxArray 进行操作的函数,而以 mex 开头的 Matlab API 函数则提供 Matlab 环境后台操作的函数。其中以 mex 开头的 Matlab API 函数只能在 MEX 文件中应用。

3.5　Matlab 与 C 语言混合编程常用的数据类型

3.5.1　size_t 类型

因为下面讨论中要用到 size_t 类型,因而首先说明一下 size_t 类型。不同的机器和编译器,size_t 的定义是不同的,比如下面这几种定义都有可能出现:

```
typedef unsigned int size_t;
typedef int size_t;
typedef short size_t;
typedef unsigned short size_t;
```

当然也可能出现其他定义。总的来说,size_t 主要用来作为 sizeof 函数的返回类型,或者作为描述变量或者内存长度的一种类型。之所以采用 size_t 而不是直接采用 int,long,short 或者其他整型,是为了程序代码的通用性,假如程序执行的平台变了,那么只要改变 size_t 的定义即可。

3.5.2 Matlab C 语言接口数据类型

Matlab C 语言接口中有几个特殊的数据类型，即：

mwIndex，mwSize，mxChar，mxLogical，mxClassID，mxComplexity

下面分别进行介绍。

1. mwIndex 和 mwSize

mwIndex 用以表示阵列索引；mwSize 用以表示大小，比如阵列维数或者阵列长度。mwIndex 和 mwSize 的定义包含在 extern\include\tmwtypes.h 文件中，定义如下：

```
# ifdef MX_COMPAT_32
  typedef int mwSize;
  typedef int mwIndex;
  typedef int mwSignedIndex;
# else
      typedef size_t      mwSize;
      typedef size_t      mwIndex;
      typedef ptrdiff_t mwSignedIndex;
    # endif
```

从这个定义可以看出，当 MX_COMPAT_32 宏定义存在时，Matlab 对 mwIndex 和 mwSize 变量的定义实际上是 int 类型；当 MX_COMPAT_32 宏定义不存在时，Matlab 对 mwIndex 和 mwSize 变量的定义实际上是 size_t 类型。

Matlab 之所以推出这两个变量而不再沿用原来表示阵列索引和阵列长度的 int 类型(Matlab 6.5 以前没有 mwIndex 和 mwSize 类型)，是为了适应 64 位处理器和处理更大规模阵列的需要(阵列长度大于 2^31－1)。编译 Matlab MEX 文件的 MEX 命令增加了一个 -largeArrayDims选项(Matlab 也推荐使用该选项)，如果编译 MEX 文件时不带此选项，此时 mwSize 和 mwIndex 定义为 int；如果编译 MEX 文件时带此选项，此时 mwSize 和 mwIndex 定义为 size_t。从 Matlab 2006b 开始，Matlab 对稀疏矩阵的存储方式与以前的版本相比发生了变化，只有使用-largeArrayDims 选项，才能正确使用操作稀疏矩阵的 mx- API 函数，所以用户编译 MEX 文件时最好使用-largeArrayDims 选项(Matlab 准备在 Matlab 2007 之后的版本中将-largeArrayDims 作为编译 MEX 文件的默认选项)。

2. mxChar 和 mxLogical

mxChar 和 mxLogical 是 Matlab 自定义的两个数据类型，在 extern\include\matrix.h 中可以找到它们的定义，即

```
# if defined(__APPLE__) && defined(__ppc__)
typedef unsigned char mxLogical;
# else
typedef bool        mxLogical;
# endif
typedef char16_t mxChar;
```

可以看出，在非苹果机和非 PowerPC 系的处理器中，mxLogical 实际上就是 bool 类型；而 mxChar 类型实际上就是 16 位字符类型（Matlab 中所有的字符都采用 16 位双字节存储）。

3. mxClassID

mxClassID 用来描述 Matlab 的阵列类型，所有的 Matlab 阵列类型都有一个对应的枚举量 ID，比如元组阵列的 ID 为 mxCELL_CLASS。因而，使用 mxClassID 类型可以很方便地判断某一 Matlab 阵列（用 mxArray 表示）属于哪一种类型。在文件 extern\include\matrix. h 中，对 mxClassID 类型的定义如下所示。

```
typedef enum {
    mxUNKNOWN_CLASS = 0,
    mxCELL_CLASS,
    mxSTRUCT_CLASS,
    mxLOGICAL_CLASS,
    mxCHAR_CLASS,
    mxVOID_CLASS,
    mxDOUBLE_CLASS,
    mxSINGLE_CLASS,
    mxINT8_CLASS,
    mxUINT8_CLASS,
    mxINT16_CLASS,
    mxUINT16_CLASS,
    mxINT32_CLASS,
    mxUINT32_CLASS,
    mxINT64_CLASS,
    mxUINT64_CLASS,
    mxFUNCTION_CLASS,
        mxOPAQUE_CLASS,
    mxOBJECT_CLASS
    } mxClassID;
```

4. mxComplexity

mxComplexity 类型用以区分浮点数值阵列是实数还是复数，它也是一个枚举类型，在文件 extern\include\matrix. h 中，对其的定义如下所示。

```
typedef enum {
    mxREAL,
    mxCOMPLEX
} mxComplexity;
```

假设用户需要创建一个 5 行 3 列的复数双精度数值阵列，则用户需要传递 mxCOMPLEX 变量给 mxCreateDoubleMatrix 函数，如下所示。

```
mxArray * pa;
pa = mxCreateDoubleMatrix(5,3,mxCOMPLEX);
```

3.6 操作 Matlab 阵列 mxArray 的 mx 函数

1. int mxAddField(mxArray * array_ptr,const char * field_name);

函数功能：

为结构体增加一个域。

参数说明：

mxArray * array_ptr 指向结构体的 mxArray 指针。

const char * field_name 要添加的域的名称。

返回值：

如果函数调用成功则返回新添加域的计数，否则返回-1。

2. char * mxArrayToString(const mxArray * array_ptr);

函数功能：

将字符阵列转换为C字符串。

参数说明：

const mxArray * array_ptr //待转换字符阵列。

返回值：

返回转换后的字符串，如果函数调用失败则返回 NULL 空指针，此函数用以转换双字节的字符串。

3. mwIndex mxCalcSingleSubscript(const mxArray * array_ptr,mwSize nsubs,mwIndex * subs);

函数功能：

计算阵列元素(subs(0),subs(1),…,subs(nsubs-1))相对于阵列元素(0,0,0)的线性偏移量。

参数说明：

const mxArray * array_ptr 输入 Matlab 阵列的 mxArray 指针。

mwSize nsubs 表示一维数组 subs 的元素个数。

mwIndex * subs 表示需要求解线性偏移量的阵列元素的索引。

返回值：

如果函数调用成功则返回一个(0,N-1)之间的数，其中 N 为输入 mxArray 阵列的元素个数，否则返回-1。

附加说明：

所有 Matlab 阵列，不论其维数大小其在内存中都是按照一维的方式存储的。与C语言数组的存储方式不同的是，Matlab 阵列在内存中是按列的方式存储的。如表 3-2 所列的二维 Matlab 字符阵列，其在内存中的存储方式和其线性索引如表 3-3 所列。

表 3-2 Matlab 2×5 字符阵列

A	B	C	D	E
F	G	H	I	J

表 3－3　Matlab 阵列的存储方式及线性索引示意图

字　符	A	F	B	G	C	H	D	I	F	J
索　引	0	1	2	3	4	5	6	7	8	9

对于多维阵列，其原理与二维阵列类似，其在内存中的存储方式均是一维线性的。只不过多维阵列相对二维阵列来说理解上可能比较困难。以三维阵列为例（实际很少用到三维以上的阵列），其字符阵列如图 3－2 所示。三维 Matlab 阵列在内存中的线性存储方式如表 3－4 所列。

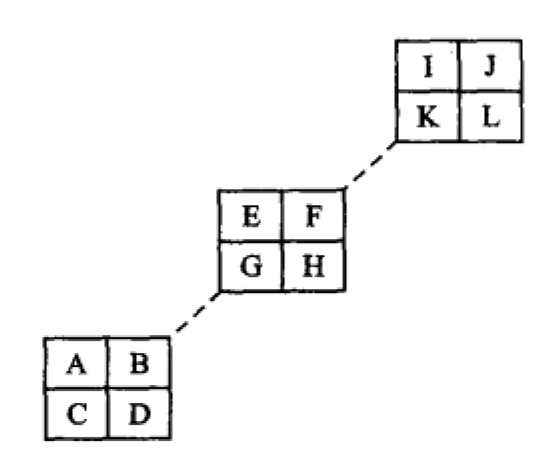


图 3－2　Matlab 2×2×3 字符阵列

mxCalcSingleSubscript 函数就是用来计算指定阵列元素相对于第一个阵列元素的线性偏移值，例如对于表 3－4所列的三维阵列，试求 Matlab 阵列元素(1,2,2)的线性偏移值，则：

```
int nsubs = 3;
int subs[3];
int nOffSet;
subs[0] = 1;
subs[1] = 2;
subs[2] = 2;
//则输出 nOffSet 为 5
nOffSet = mxCalcSingleSubscript(arrayPointer,nsubs,subs);
```

另外需要注意的是：Matlab 阵列的索引都是从 1 开始的，而 C 语言数组的索引都是从 0 开始的，因此，Matlab 阵列元素(1,2,2)相当于 C 语言数组的索引(0,1,1)。

表 3－4　Matlab 阵列的存储方式及线性索引示意

字　符	A	C	B	D	E	F	G	H	I	J	K	L
索　引	0	1	2	3	4	5	6	7	8	9	10	11

4. void * mxCalloc(size_t n,size_t size);

函数功能：

用 Matlab 内存管理器动态分配内存。

参数说明：

size_t n　动态分配的内存单元的个数。

size_t size　动态分配的内存单元的大小。

返回值：

如果内存分配成功则返回分配的内存的首地址的指针，如果失败则返回 NULL；但是如果在 MEX 文件的执行过程中如果内存分配失败的话，则中止 MEX 文件的执行。

附加说明：

在动态分配 Matlab 应用相关的内存时(如 mxArray),必须使用 mxCalloc 或其他 Matlab 内存分配函数动态动态分配内存。在 MEX 文件中,Matlab 的内存管理器会自动维护所有由 mxCalloc 分配的内存,并在 mex 程序返回 Matlab 命令行时自动释放由 mxCalloc 分配的内存。

在应用 Matlab API 的独立可执行的程序中,mxCalloc 在默认的情况下调用标准 C 的 calloc 函数,用户也可以用 mxSetAllocFcns 注册自己的内存分配函数。

5. mxArray * mxCreateCellArray(mwSize ndim,const mwSize * dims);

函数功能:

创建 N 维空的 Matlab 元组阵列。

参数说明:

mwSize ndim　一维数组 dims 的元素个数。

mwSize * dims　表示 Cell 阵列各维大小的一维数组,其 dims[0]代表 Cell 阵列的第 1 维的大小(行数)。假如要创建 3×4×5 的 Cell 阵列,则:

```
mwSize ndim = 3,
mwSize dims[3] = {3,4,5};
mxArray * cell345;
cell345 = mxCreateCellArray(ndim,dims);
```

返回值:

如果函数调用成功,返回创建的 Cell 阵列的 mxArray 的指针;如果函数调用失败,返回 NULL。

6. mxArray * mxCreateCellMatrix(mwSize m,mwSize n);

函数功能:

创建 m 行 n 列 Cell 矩阵。

参数说明:

mwSize m　待创建 Cell 矩阵的行数。

mwSize n　待创建 Cell 矩阵的列数。

返回值:

返回 m×n Cell 阵列的 mxArray 指针。

7. mxArray * mxCreateCharArray(mwSize ndim,const mwSize * dims);

函数功能:

创建字符阵列。

参数说明:

mwSize ndim　待创建字符阵列的维数,ndim 必须大于零。由于 Matlab 中标量用 1×1 的阵列,因而 nidm 应该大于等于 2。

mwSize * dims　待创建字符阵列的各维大小,整型数组 dims 的维数应该大于等于 2。

返回值:

返回创建的字符阵列,如果失败则返回 NULL。

8. mxArray * mxCreateCharMatrixFromStrings(mwSize m,const char * * str);

函数功能：

创建二维字符阵列，其初值为输入的字符串。

参数说明：

mwSize m　输入的 C 语言字符串的个数，即创建的字符矩阵的行数。

const char * * str　输入的 C 语言字符串指针数组。

返回值：

如果函数调用成功，则返回创建的二维字符阵列的 mxArray 指针；否则返回 NULL。

附加说明：

用这种方式创建的二维数组，其行数由 m 指定，其列数由输入的 m 个字符串的最大长度决定。另外需要说明的是，在 Matlab 字符阵列中，字符类型为 mxChar，而不是 char。mxChar 的定义如下：

```
typedef char16_t mxChar; //16 位字符，而不是 C 语言中常用的 8 位字符
```

通过下面例子，读者可以大致了解 mxCreateCharMatrixFromStrings 函数的使用方法。

```
/* mxCreateCharMatrixFromString.c 文件内容 */
#include "mex.h"
#include "matrix.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    char s1[] = "Bei Jing";
    char s2[] = "Shang Hai";
    char s3[] = "Tian Jin";
    char s4[] = "Chong Qing";
    char * s[4] = {s1,s2,s3,s4};
    mxArray * charArray;
    /*
        创建一个 4 x 10 字符矩阵，其内容如下：
        Bei Jing
        Shang Hai
        Tian Jin
        Chong Qing
    */
    charArray = mxCreateCharMatrixFromStrings(4,s);
    if(nlhs == 1)
    {
        plhs[0] = charArray;
    }
    else
    {
        mexPrintf("输出阵列的个数错误！\n");
```

```
mxDestroyArray(charArray);
}
}
```

9. mxArray * mxCreateDoubleMatrix(mwSize m,mwSize n, mxComplexity ComplexFlag);

函数功能:

创建双精度矩阵(m×n,m>=1,n>=1)。

参数说明:

mwSize m　数值矩阵的行数。

mwSize n　数值矩阵的列数。

mxComplexity ComplexFlag　数值阵列的类型,其中 mxComplexity 的定义如下:

```
typedef enum mxComplexity {mxREAL = 0,mxCOMPLEX};
```

mxComplexity 同样是一个枚举类型,其中 mxREAL 表示当前 mxArray 是一个实数数组,没有虚部。mxCOMPLEX 表示当前 mxArray 是一个复数数组,存在虚部,即 GetPi 返回不为空。

返回值:

返回生成的二维双精度数值阵列的 mxArray 指针。

10. mxArray * mxCreateDoubleScalar(double value);

函数功能:

创建双精度标量(1×1),与 mxCreateScalarDouble 函数的功能相同。mxCreateDoubleScalar 函数的功能于下面的语句相同,只不过书写起来比较简洁。

```
pa = mxCreateDoubleMatrix(1,1,mxREAL);
* mxGetPr(pa) = value;
```

参数说明:

double value　表示生成的双精度标量的值。

返回值:

返回生成的 1×1 双精度数值阵列。

11. mxArray * mxCreateLogicalArray(mwSize ndim,const mwSize * dims);

函数功能:

创建 Matlab 逻辑阵列。

参数说明:

mwSize ndim　待创建逻辑阵列的维数,ndim 必须大于零。由于 Matlab 中标量用 1×1 的阵列,因而 nidm 应该大于等于 2。

mwSize * dims　待创建逻辑阵列的各维大小,整型数组 dims 的维数应该大于等于 2。

返回值:

返回生成的 Matlab 逻辑阵列的 mxArray 指针。

12. mxArray * mxCreateLogicalMatrix(mwSize m,mwSize n);

函数功能：

创建 Matlab 逻辑矩阵。与 mxCreateLogicalArray 不同，mxCreateLogicalMatrix 只创建二维的 Matlab 逻辑阵列。

参数说明：

mwSize m　待创建的二维的 Matlab 逻辑阵列的行数。

mwSize n　待创建的二维的 Matlab 逻辑阵列的列数。

返回值：

创建的二维 Matlab 逻辑阵列的 mxArray 指针。

13. mxArray * mxCreateLogicalScalar(mxLogical value);

函数功能：

创建的 Matlab 逻辑型标量。

参数说明：

mxLogical value　创建的 Matlab 逻辑型标量的初值，其中 mxLogical 的定义如下：

```
typedef bool mxLogical
```

返回值：

创建的 Matlab 逻辑型标量阵列的 mxArray 指针。

附加说明：

mxCreateLogicalScalar 函数的功能和下面语句的功能是相同的，只不过 mxCreateLogicalScalar 函数书写起来比较简洁。

```
mxArray * pa = mxCreateLogicalMatrix(1,1);
* mxGetLogicals(pa) = value;
```

14. mxArray * mxCreateNumericArray(mwSize ndim,const mwSize * dims,mxClassID class,mxComplexity,ComplexFlag);

函数功能：

创建数值阵列。

参数说明：

mwSize ndim　待创建数值阵列的维数。

const mwSize * dims　待创建数值阵列各维的大小，其中 dims 是一个一维数组，数组的大小为 ndim。

mxClassID class　表示创建的数值阵列的类型；mxClassID 是一个枚举类型，用以表示 Matlab 开发环境中当前存在的所有 Matlab 数组类型。通过函数 mxGetClassID 可以得到 mxArray 的数组类型。

mxComplexity ComplexFlag　表示数值阵列的是实数还是复数，具体说明见 3.5.2 节。

返回值：

创建的 Matlab 数值阵列的 mxArray 指针。

15. mxArray * mxCreateNumericMatrix(mwSize m,mwSize n,
mxClassID class,mxComplexity ComplexFlag);

函数功能：

创建数值矩阵。

参数说明：

mwSize m　数值矩阵的行数。

mwSize n　数值矩阵的列数。

mxClassID class　数值矩阵的类型,见 mxCreateNumericArray 关于 mxClassID 的说明;

mxComplexity ComplexFlag 表示数值阵列的是实数还是复数,具体说明见 3.5.2 节。

返回值：

创建的 Matlab 数值矩阵的 mxArray 指针。

16. mxArray * mxCreateScalarDouble(double value);

函数功能：

创建数值标量。

参数说明：

double value　创建的数值标量的初始值。

返回值：

创建的 Matlab 双精度标量阵列的 mxArray 指针。

17. mxArray * mxCreateSparse(mwSize m,mwSize n,mwSize nzmax,
mxComplexity ComplexFlag);

函数功能：

创建稀疏矩阵。

参数说明：

mwSize m　稀疏矩阵的行数。

mwSize n　稀疏矩阵的列数。

mwSize nzmax　最大的非零元素个数,其中 nzmax 要不大于 m * n 的值。

mxComplexity complexFlag　待创建的稀疏矩阵的实型或复型标志,其中 complexFlag 如果为 mxREAL 表示创建的稀疏矩阵为实型;如果为 mxCOMPLEX,则表示创建的稀疏矩阵为复型。

返回值：

创建 Matlab 稀疏矩阵的 mxArray 指针。

18. mxArray * mxCreateSparseLogicalMatrix(mwSize m,mwSize n);

函数功能：

创建逻辑型的稀疏矩阵。

参数说明：

mwSize m　逻辑型稀疏矩阵的行数。

mwSize n　逻辑型稀疏矩阵的列数。

返回值：

创建 Matlab 逻辑型稀疏矩阵的 mxArray 指针。

附加说明：

由于逻辑型变量只有 true(1)和 flase(0)两种情况，因而对于 false(0)占大部分比例并且很大的逻辑型矩阵，可以通过使用稀疏矩阵的办法达到降低占用内存空间的目的。

19. mxArray * mxCreateString(const char * str);

函数功能：

创建 Matlab 字符型阵列，并将根据输入的字符串为其赋初值。

参数说明：

const char * str 用做初始化待创建 Matlab 字符阵列的字符串。

返回值：

创建的 Matlab 字符类型阵列的 mxArray 指针。

20. mxArray * mxCreateStructArray(mwSize ndim, const mwSize * dims, int nfields, const char * * field_names);

函数功能：

创建 Matlab 结构体阵列。

参数说明：

mwSize ndim 待创建结构体阵列的维数，ndim 必须大于零。由于 Matlab 中标量用 1×1 的阵列，因而 nidm 应该大于等于 2。

mwSize * dims 待创建结构体阵列的各维大小，整型数组 dims 的维数应该大于等于 2。

int nfileds 结构体阵列的域的个数。

char * * field_names 结构体阵列的域名字符串数组。

返回值：

创建的 Matlab 结构体阵列 mxArray 指针。

21. mxArray * mxCreateStructMatrix(mwSize m, mwSize n, int nfields, const char * * field_names);

函数功能：

创建 Matlab 结构体矩阵。

参数说明：

mwSize m 结构体矩阵的行数。

mwSize n 结构体矩阵的列数。

int nfields 结构体阵列的域的个数。

char * * field_name 结构体阵列的域名字符串数组。

返回值：

创建的 Matlab 结构体阵列的 mxArray 指针。

22. void mxDestroyArray(mxArray * array_ptr);

函数功能：

释放由 mxCreate * * * 函数创建的 mxArray 使用的内存。

参数说明：

mxArray * array_ptr 需要释放的 mxArray 指针。

23. mxArray * mxDuplicateArray(const mxArray * in);

函数功能:

复制任何一个 mxArray 变量。

参数说明:

mxArray * in 需要复制的 mxArray 的指针。

返回值:

复制的 Matlab 阵列 mxArray 指针。

24. void mxFree(void * ptr);

函数功能:

释放由 mxCalloc 分配的内存。

参数说明:

void * ptr 指向由 mxCalloc 分配的内存的首地址的指针。

25. mxArray * mxGetCell(const mxArray * array_ptr, mwIndex index);

函数功能:

得到 Matlab Cell 阵列中索引为 index Cell 元素里保存的 Matlab 阵列。

参数说明:

const mxArray * array_ptr Matlab Cell 阵列的 mxArray 指针。

mwIndex index 从 Cell 阵列第一个元素到需要得到其内容的 Cell 阵列元素的线性偏移量,可以由 mxCalcSingleSubscript 函数得到。

返回值:

索引为 index 的 Cell 元素的 mxArray 指针。

26. mxChar * mxGetChars(const mxArray * array_ptr);

函数功能:

得到 Matlab 字符类型阵列字符数据的指针。

27. mxClassID mxGetClassID(const mxArray * array_ptr);

函数功能:

得到以 mxClassID 枚举类型表示的 Matlab 阵列类型。

28. const char * mxGetClassName(const mxArray * array_ptr);

函数功能:

得到以字符串表示的 Matlab 阵列类型。

29. void * mxGetData(const mxArray * array_ptr);

函数功能:

得到非 double 型数值阵列的数据指针,double 型数值阵列的数据指针用 mxGetPr 和 mxGetPi 得到。

30. const mwSize * mxGetDimensions(const mxArray * array_ptr);

函数功能:

和 mxGetNumberOfDimensions 联合使用，用来确定输入 Matlab 阵列的维数以及各维的大小。

返回值：

函数返回一个一维整型数组，数组的大小由 mxGetNumberOfDimensions 来确定。

31. size_t mxGetElementSize(const mxArray * array_ptr);

函数功能：

得到 Matlab 阵列的单个元素需要的存储空间(bytes)。例如：

```
mxArray *  a;
size_t aESize;
a = mxCreateNumericArray(NDIMS,dims,mxUINT16_CLASS,mxREAL);
aESize = mxGetElementSize(a);
```

32. double mxGetEps(void);

函数功能：

返回 Matlab 所能表示的最小浮点数，即对于如果两个变量 a 和 b，如果 abs(a－b)＞eps，则可以认为 a 和 b 是不相等的；否则则认为 a 等于 b。

33. mxArray * mxGetField(const mxArray * array_ptr,mwIndex index, const char * field_name);

函数功能：

给定阵列元素的线性索引和域名，得到相应域值。

参数说明：

const mxArray * array_ptr　Matlab 结构体阵列。

mwIndex index　Matlab 结构体阵列索引。

char * fileld_name　结构体域名。

返回值：

返回相应的域值。

34. mxArray * mxGetFieldByNumber(const mxArray * array_ptr, mwIndex index,int field_number);

函数功能：

给定阵列元素的线性索引和域号，得到相应的域值。

参数说明：

const mxArray * array_ptr　Matlab 结构体阵列。

mwIndex index　Matlab 结构体阵列索引。

int field_number　结构体域号，结构体第一个域的域号为 0，第二个域的域号为 1，以此类推。域号的值不能超过 N－1，N 为结构体域的最大个数。

返回值：

返回相应的域值。

35. const char * mxGetFieldNameByNumber(const mxArray * array_ptr,int field_number);

函数功能:

根据结构体域的域号得到相应的域名。

参数说明:

const mxArray * array_ptr Matlab 结构体阵列。

int field_number Matlab 结构体域号。

返回值:

域号为 field_number 对应的 Matlab 域名。

36. int mxGetFieldNumber(const mxArray * array_ptr,const char * field_name);

函数功能:

根据结构体域的域名得到相应的域号。

参数说明:

const mxArray * array_ptr Matlab 结构体阵列。

char * field_name Matlab 结构体阵列域名。

返回值:

返回结构体域域号。

37. void * mxGetImagData(const mxArray * array_ptr);

函数功能:

得到 Matlab 非双精度型数值阵列的虚部数据指针。

参数说明:

const mxArray * array_ptr Matlab 数值阵列的 mxArray 指针。

38. double mxGetInf(void);

函数功能:

得到 Matlab 的正无穷大,有些操作本身就会返回无穷大,例如 5/0 或者 exp(1000)等。

39. mwIndex * mxGetIr(const mxArray * array_ptr);

函数功能:

得到 Matlab 稀疏矩阵的 ir 数据指针。

参数说明:

const mxArray * array_ptr Matlab 稀疏矩阵 mxArray 指针。

返回值:

Matlab 稀疏矩阵的 ir 数据指针。

附加说明:

ir 指针指向稀疏矩阵非零数据的行值数据,这些行值数据和 jc 指针指向的列值数据联合使用,可以确定 Matlab 稀疏矩阵的非零元素的行列索引值。ir 指针指向的一位数组的大小为 nzmax,其中 nzmax 为 Matlab 稀疏矩阵的最大非零元素的个数。

40. mwIndex * mxGetJc(const mxArray * array_ptr);

函数功能：

得到 Matlab 稀疏矩阵的 jc 数据指针。

参数说明：

const mxArray *　array_ptr　Matlab 稀疏矩阵 mxArray 指针。

返回值：

Matlab 稀疏矩阵的 jc 数据指针。

附加说明：

jc 指针指向一个整型数组，数组的长度为 n+1，其中 n 为 Matlab 稀疏矩阵的列数。jc 指针指向的整型数组的数据含义为：(jc[i+1]-jc[i], i<n)表示稀疏矩阵第 i 列数据所包含的非零元素的个数，因而 jc 指向的整型数组和 ir 指向的整型数组联合使用可以确定 Matlab 稀疏矩阵的非零元素的行列索引值。

41. mxLogical * mxGetLogicals(const mxArray * array_ptr);

函数功能：

得到 Matlab 逻辑型阵列的数据指针。

参数说明：

const mxArray *　array_ptr　Matlab 逻辑型阵列。

返回值：

Matlab 逻辑型阵列的数据指针。

42. size_t mxGetM(const mxArray * array_ptr);

函数功能：

得到 Matlab 阵列的行数。

43. size_t mxGetN(const mxArray * array_ptr);

函数功能：

得到 Matlab 阵列的列数。

44. double mxGetNaN(void);

函数功能：

得到 NaN 的值(NaN 是 Not a Number 的简写)。

45. mwSize mxGetNumberOfDimensions(const mxArray * array_ptr);

函数功能：

得到 Matlab 阵列的维数。

46. mwSize mxGetNumberOfElements(const mxArray * array_ptr);

函数功能：

得到 Matlab 阵列的元素个数。

47. int mxGetNumberOfFields(const mxArray * array_ptr);

函数功能：

得到 Matlab 结构体阵列的域的个数。

参数说明:

const mxArray * array_ptr　Matlab 结构体阵列的 mxArray 的指针。

48. mwSize mxGetNzmax(const mxArray * array_ptr);

函数功能:

得到 Matlab 稀疏矩阵的最大非零元素个数(nzmax)的值。

49. double * mxGetPi(const mxArray * array_ptr);

函数功能:

得到 Matlab 双精度数值阵列的虚部数据指针。

50. double * mxGetPr(const mxArray * array_ptr);

函数功能:

得到 Matlab 双精度数值阵列的实部数据指针。

51. double mxGetScalar(const mxArray * array_ptr);

函数功能:

得到 Matlab 标量阵列的值。

52. int mxGetString(const mxArray * array_ptr, char * buff, int buflen);

函数功能:

得到 Matlab 字符类型阵列的字符数据。此时得到的字符数据不是 Matlab 的 mxChar 类型的,而是 C 语言 char 类型的,并且是以"\0" 字符结束的。

参数说明:

const mxArray * array_ptr　Matlab 字符阵列的 mxArray 指针。

char * buff　接受 Matlab 字符阵列字符串的 C 字符串指针。

int bufflen　一维 buff 字符数组的大小,包括"\0"C 语言字符串结束符在内。

返回值:

如果函数调用成功,并且 buff 的大小足够接受 Matlab 字符阵列的所有字符,则返回 0,否则返回 1。

附加说明:

如果输入的 Matlab 字符阵列是多维的,此时将会按照一维的方式返回字符阵列中所有的字符。这时候,返回的字符串是 Matlab 字符阵列按列存储的结果。例如,['ABCD';'EFGH'] 返回的是字符串"AEBFCGDH",下面就是一个利用 mxGetString 函数显示输入字符阵列内容的 MEX 文件的例子。

```
/ * mxGetString.c 文件内容 * /
# include "mex.h"
# include "matrix.h"
# ifndef NULL
# define NULL 0
# endif
```

```
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,
                 const mxArray * prhs[])
{
    int i=0;
    mwSize j=0;
    char * buff;
    int nLen;
    mwSize ndims;
    const mwSize * dims;

    unsigned int nbufflen;
    int status;
    if(nrhs<=0)
    {
        mexPrintf("输入参数个数为零！\n");
        return;
    }
    for(i=0;i<nrhs;i++)
    {
        ndims=mxGetNumberOfDimensions(prhs[i]);
        dims=mxGetDimensions(prhs[i]);
        nbufflen=1;
        for(j=0;j<ndims;j++)
        {
            nbufflen *= dims[j];
        }
        nbufflen +=1;//用于存放结束字符'\0'
        if(! mxIsChar(prhs[i]))
        {
            mexPrintf("第%d个输入的参数不是字符数组！\n",i+1);
            continue;
        }

        buff=(char *) mxCalloc(nbufflen,sizeof(char));
        status=mxGetString(prhs[i],buff,nbufflen);
        if(status != 0)
        {
            mexWarnMsgTxt("字符串接收空间不足！\n");
        }
        mexPrintf("字符串%d:%s\n",i+1,buff);
        mxFree(buff);
        buff=NULL;
    }
```

}

53. bool mxIsCell(const mxArray * array_ptr);

函数功能:

判断输入 Matlab 阵列是否为 Cell 阵列。

54. bool mxIsChar(const mxArray * array_ptr);

函数功能:

判断输入 Matlab 阵列是为否字符阵列。

55. bool mxIsClass(const mxArray * array_ptr,const char * name);

函数功能:

判断输入 Matlab 阵列是否为变量 name 标志的类型。

参数说明:

const mxArray * array_ptr　Matlab 阵列 mxArray 指针。

const char * name　标志 Matlab 阵列类型的字符串,其中 name 字符串和 mxClassID 的对应关系如表 3-5 所列。

表 3-5　mxClassID 与及其相对应的字符串

名　称	对应的 mxClassID
cell	mxCELL_CLASS
char	mxCHAR_CLASS
double	mxDOUBLE_CLASS
function handle	mxFUNCTION_CLASS
int8	mxINT8_CLASS
int16	mxINT16_CLASS
int32	mxINT32_CLASS
logical	mxLOGICAL_CLASS
single	mxSINGLE_CLASS
struct	mxSTRUCT_CLASS
uint8	mxUINT8_CLASS
uint16	mxUINT16_CLASS
uint32	mxUINT32_CLASS
unknown	mxUNKNOWN_CLASS

附加说明:

下面几种函数的调用方式效果是相同的,即

- mxIsClass("double");
- mxIsDouble(array_ptr);
- strcmp(mxGetClassName(array_ptr),"double");

56. bool mxIsComplex(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为复数型阵列。

57. bool mxIsDouble(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列是否为为双精度型阵列。

58. bool mxIsEmpty(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列是否为空，Matlab 阵列为空的条件是其各维的大小均为 0。

59. bool mxIsFinite(double value);

函数功能：

判断输入的双精度数值是否为有限值。

60. bool mxIsInf(double value);

函数功能：

判断输入的双精度数值是否为无穷大。

61. bool mxIsInt8(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 8 位有符号整型。

62. bool mxIsInt16(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 16 位有符号整型。

63. bool mxIsInt32(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 32 位有符号整型。

64. bool mxIsUint8(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 8 位无符号整型。

65. bool mxIsUint16(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 16 位无符号整型。

66. bool mxIsUint32(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为 32 位无符号整型。

67. bool mxIsLogical(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 阵列类型是否为逻辑型。

68. bool mxIsLogicalScalar(const mxArray * array_ptr);

函数功能：

判断输入的 Matlab 标量阵列类型是否为逻辑型。

69. bool mxIsLogicalScalarTrue(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 逻辑型标量的值是否为 true。

70. bool mxIsNaN(double value);

函数功能:

判断输入的 double 值是否为 NaN(Not a Number)。

71. bool mxIsNumeric(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为数值阵列。

72. bool mxIsSingle(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为单精度数值阵列。

73. bool mxIsSparse(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为稀疏矩阵。

74. bool mxIsStruct(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为结构体阵列。

75. bool mxIsIntX(const mxArray * array_ptr);

函数功能:

判断输入的 Matlab 阵列是否为 X 位整型阵列,其中 X 为 8、16 或者 32。

76. void * mxMalloc(size_t n);

函数功能:

采用 Matlab 内存管理模块动态分配内存,与 C 语言的 malloc 类似。在 MEX 文件中,Matlab 内存管理模块会自动维护一个由 mxMalloc 分配的内存列表,当 MEX 文件执行返回到 Matlab 命令行时,Matlab 内存管理模块会自动释放由 mxMalloc 分配的内存。但是在使用此函数的 Matlab 与 C 语言混编独立可执行文件中,mxMalloc 在默认情况下调用 C 语言的 malloc 函数,开发人员可以通过函数 mxSetAllocFcns 设置自己的内存分配函数。

77. void * mxRealloc(void * ptr, size_t size);

函数功能:

重新分配内存块的大小,采用此函数有可能会造成内存泄漏。

78. extern void mxRemoveField(mxArray * array_ptr, int field_number);

函数功能:

去除结构体阵列的一个域。

参数说明:

mxArray * array_ptr 结构体阵列的 mxArray 指针。
int field_number 域号。

79. void mxSetAllocFcns(calloc_proc callocfcn, free_proc freefcn, realloc_proc reallocfcn, malloc_proc mallocfcn);

函数功能:

注册自己开发的 Matlab 内存分配和释放函数。

参数说明:

calloc_proc callocfcn 手动注册的 calloc 函数名。
free_proc freefcn 手动注册的 free 函数名。
realloc_proc reallocfcn 手动注册的 realloc 函数名。
malloc_proc mallocfcn 手动注册的 malloc 函数名。

附加说明:

在 MEX 文件中,不允许调用 mxSetAllocFcns,否则会导致编译出错。在独立可执行的调用 mx 函数的 C 语言程序中,默认情况下 mxAlloc,mxRealloc 和 mxMalloc 只是分别调用 C 语言的 alloc,realloc 和 malloc 函数,而 mxFree 则在输入参数非 NULL 的情况下调用 C 语言 free 函数。在独立可执行调用 mx 函数的 C 语言程序中,调用 mxSetAllocFcns 可以注册用户自己的内存分配和释放函数。

手动注册的 callocfcn 函数必须按照下面的格式声明:

void * callocfcn(size_t nmemb, size_t size);

手动注册的 freefcn 函数必须按照下面的格式声明:

void freefcn(void * ptr);

手动注册的 reallocfcn 函数必须按照下面的格式声明:

void * reallocfcn(void * ptr, size_t size);

手动注册的 mallocfcn 函数必须按照下面的格式声明:

void * mallocfcn(size_t n);

程序实例:

下面就是一个手动注册用户自己的 calloc,free,malloc 和 realloc 函数的例子。

```
/* setalloc.c 文件内容 */
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include "matrix.h"

/* 用户注册的 calloc 函数 */
void * my_calloc(size_t n, size_t  size)
{
    void   * ptr;
    printf("\t 调用注册的 calloc 函数! \n");
    if ((n <= 0) || (size <= 0))
    {
```

```
            return NULL;
        }
        else
        {
            ptr = calloc(n,size);
            return(ptr);
        }
}
/* 用户注册的 free 函数 */
void my_free(void * ptr)
{
        printf("\t 调用注册的 free 函数! \n");
        if (ptr == NULL)
        {
            return;
        }
        else
        {
            free(ptr);
        }
}

/* 用户注册的 realloc 函数 */
void * my_realloc(void * ptr,size_t  size)
{
        printf("\t 调用注册的 realloc 函数! \n");
        if (size == 0)
        {
            free(ptr);
        }
        if (size <= 0)
        {
            return NULL;
        }
        if (ptr == NULL)
        {
            ptr = malloc(size);
        }
        else
        {
            ptr = realloc(ptr,size);
        }
        return(ptr);
}
```

```
/* 用户注册的 malloc 函数 */
void * my_malloc(size_t  size)
{
  void   * ptr;
  printf("\t 调用注册的 malloc 函数! \n");
  if (size <= 0)
  {
      return NULL;
  }
  else
  {
      ptr = malloc(size);
      return(ptr);
  }
}

/* 测试函数 */
int main(int argc,char *  argv[])
{
    char * x;
    mxArray * pa;
    mxSetAllocFcns(my_calloc,my_free,my_realloc,my_malloc);
    printf("创建一个字符阵列...\n");
    pa=mxCreateString("手动注册内存分配函数的实例! \n");

    printf("\n 调用 mxCalloc 函数为字符串分配内存...\n");
    x=(char *)mxCalloc(255,sizeof(char));
    mxGetString(pa,x,255);
    printf("\n 打印字符串中的内容:\n\t %s\n",x);
    printf("\n 释放分配的内存...\n");
    mxFree(x);
    printf("\n 释放字符阵列...");
    mxDestroyArray(pa);
    getchar();
    return 0;
}
```

执行 mbuild setalloc.c 将上述 C 语言程序编译为可执行文件，运行测试函数的执行结果如下所示。

```
创建一个字符阵列...
    调用注册的 calloc 函数!
    调用注册的 calloc 函数!
```

```
调用 mxCalloc 函数为字符串分配内存...
    调用注册的 calloc 函数!
打印字符串中的内容:
    手动注册内存分配函数的实例!
释放分配的内存...
    调用注册的 free 函数!
释放字符阵列...调用注册的 free 函数!
    调用注册的 free 函数!
```

80. void mxSetCell(mxArray * array_ptr,mwIndex index,mxArray * value);

函数功能:

设置 Matlab Cell 阵列的元素值。

参数说明:

mxArray * array_ptr　Matlab Cell 阵列。

mwIndex index　待设置的元组元素线性索引,用 mxCalcSingleSubscript 函数得到。

mxArray * value　待设置的元组元素值的 mxArray 指针。

附加说明:

mxSetCell 既可以为空的 Matlab 元组阵列设置值,也可以替换元组中原有的值,但是这时候需要先调用 mxDestroyArray 函数释放元组中原有元素占用的内存。

81. void mxSetData(mxArray * array_ptr,void * data_ptr);

函数功能:

设置 Matlab 非 double 型数值阵列的数据指针。

参数说明:

mxArray * array_ptr Matlab　非 double 型数值阵列。

void * data_ptr　指向数据区的指针。

附加说明:

mxSetData 与 mxSetPr 的函数功能相似,只不过 mxSetPr 用来设置 double 型数值阵列的实部数据指针。

82. int mxSetDimensions(mxArray * array_ptr,const mwSize * dims, mwSize ndim);

函数功能:

改变 Matlab 阵列的维数和各维指针。

参数说明:

mxArray * array_ptr　Matlab 阵列的 mxArray 指针。

const mwSize * dims　Matlab 各维大小的整型数组,数组大小为 ndim。

mwSize ndim　dims 数组的大小,也即 Matlab 阵列的维数。

83. void mxSetField(mxArray * array_ptr,mwIndex index,const char * field_name,mxArray * value);

函数功能:

改变 Matlab 结构体阵列的域值。

参数说明：

mxArray * array_ptr　Matlab 结构体阵列。

mwIndex index　要改变的结构体阵列元素的线性索引，用 mxCalcSingleSubscript 函数得到。

const char * field_name　要改变域值的结构体域名。

mxArray * value　待设置的 Matlab 阵列的 mxArray 指针。

附加说明：

mxSetField(pa,index,"field_name",new_value_pa)；与下面的语句的功能相同：

```
field_num=mxGetFieldNumber(pa,"field_name");
mxSetFieldByNumber(pa,index,field_num,new_value_pa);
```

84. void mxSetFieldByNumber(mxArray * array_ptr,mwIndex index, int field_number,mxArray * value);

函数功能：

与 mxSetField 函数的功能相同，只是此函数根据输入的结构体域号而不是域名来判断要更改的 Matlab 结构体域。

85. void mxSetImagData(mxArray * array_ptr,void * pi);

函数功能：

设置非 double 型数值阵列的虚部数据指针。

参数说明：

mxArray * array_ptr　Matlab 非 double 型数值阵列的 mxArray 指针。

void * pi　待设置的虚部数据指针。

附加说明：

mxSetImagData 与 mxSetPi 的函数功能相似，只不过 mxSetPi 用来设置 double 型数值阵列的实部数据指针。

86. void mxSetIr(mxArray * array_ptr,mwIndex * ir);

函数功能：

设置 Matlab 稀疏矩阵阵列的非零元素的列数信息。

参数说明：

mxArray * array_ptr　Matlab 稀疏矩阵阵列的 mxArray 指针。

mwIndex * ir　Matlab 稀疏矩阵的行信息整型数组，数组的长度为 nzmax，其中 nzmax 为 Matlab 稀疏矩阵的最大非零元素各数。

附加说明：

参考本章 3.9 稀疏数组阵列(Sparse Array)一节。

87. void mxSetJc(mxArray * array_ptr,mwIndex * jc);

函数功能：

设置 Matlab 稀疏矩阵阵列的非零元素的列数信息。

参数说明：

mxArray * array_ptr　Matlab 稀疏矩阵阵列的 mxArray 指针。

mwIndex * jc　Matlab 稀疏矩阵阵列的列信息整型数组,数组的长度为 N+1,其中 N 为 Matlab 稀疏矩阵的列数。

附加说明:

参考本章 3.9 稀疏数组阵列(Sparse Array)一节。

88. void mxSetM(mxArray * array_ptr,mwSize m);

函数功能:

设置 Matlab 阵列的行数。

参数说明:

mxArray * array_ptr　Matlab 阵列的 mxArray 指针。

int m　Matlab 阵列的行数。

89. void mxSetN(mxArray * array_ptr,mwSize n);

函数功能:

设置 Matlab 阵列的列数。

参数说明:

mxArray * array_ptr　Matlab 阵列的 mxArray 指针。

int m　Matlab 阵列的列数。

90. void mxSetNzmax(mxArray * array_ptr,mwSize nzmax);

函数功能:

设置 Matlab 稀疏矩阵阵列的最大非零元素个数。

参数说明:

mxArray * array_ptr　Matlab 稀疏矩阵阵列的 mxArray 指针。

mwSize nzmax　Matlab 稀疏矩阵阵列的最大非零元素个数。

91. void mxSetPi(mxArray * array_ptr,double * pi);

函数功能:

设置 double 型数值阵列的虚部数据指针。

参数说明:

mxArray * array_ptr　Matlab double 型数值阵列的 mxArray 指针。

double * pi　待设置的虚部数据指针。

92. void mxSetPr(mxArray * array_ptr,double * pr);

函数功能:

设置 double 型数值阵列的实部数据指针。

参数说明:

mxArray * array_ptr　Matlab double 型数值阵列的 mxArray 指针。

double * pr　待设置的实部数据指针。

3.7 Matlab mex 函数

1. int mexAtExit(void (* ExitFcn)(void));

函数功能:

注册 MEX 文件被清除或者 Matlab 关闭时调用的函数。

参数说明:

void (* ExitFcn)(void)　MEX 文件被清除或者 Matlab 关闭时调用的函数。

附加说明:

当 MEX 文件被清除,或者 Matlab 退出时,借助设置的 ExitFcn 函数可以及时清除 MEX 文件用 void mexMakeMemoryPersistent 函数设置的、MEX 文件执行完毕后依然没有被清除的内存域。另外,ExitFcn 函数也可以用来关闭全局文件或者全局 socket 链接。如果一个 MEX 文件中多次调用 mexAtExit,只有最近被设置的那个函数起作用。

```
/* mexatexit.c 文件内容 */
#include <stdio.h>
#include "mex.h"

/* 静态全局变量,MEX 文件执行退出时仍然存在! */
static FILE    *fp=NULL;

/* 当 MEX 文件被清除或者 Matlab 退出时调用此函数 */
static void CloseFile(void)
{
        mexPrintf("关闭\"文件:mydata.dat\"!\n");
        fclose(fp);
}
/*
功能:
     打开文件 mydata.dat,并向其写入一个字符串
     当 MEX 文件 mexAtExit 被清除时,关闭文件 mydata.dat
*/
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{

     char *str;

     /* 检查输入有效性 */
     if (nrhs != 1)
     {
     mexErrMsgTxt("只能有一个输入参数!\n");
     }
     if(nlhs > 1)
```

```
    {
    mexErrMsgTxt("输出参数过多! \n");
    }
    if (! (mxIsChar(prhs[0])))
    {
    mexErrMsgTxt("输入参数只能是字符类型阵列! \n.");
    }

    if (fp==NULL)
    {
        fp=fopen("mydata.dat","w");
        if (fp==NULL)
        {
            mexErrMsgTxt("打开文件 mydata.dat 出错! \n");
        }

        /* 注册 MEX 文件退出函数 */
        mexAtExit(CloseFile);
    }

    str=mxArrayToString(prhs[0]);
    if (fprintf(fp,"%s\n",str) != strlen(str) +1)
    {
        mxFree(str);
        mexErrMsgTxt("写入文件 mydata.dat 出错! \n");
    }
    mexPrintf("字符串:%s 写入到文件 mydata.dat 中! \n",str);
    mxFree(str);
}
```

该程序的执行结果如下所述。

```
>>mex mexatexit.c
>> mexatexit('退出 MEX 文件 mexatexit! ');
字符串:退出 MEX 文件 mexatexit! 写入到文件 mydata.dat 中!
>> clear all
关闭"文件:mydata.dat"!
>>
```

2. int mexCallMatlab(
int nlhs,mxArray * plhs[],
int nrhs,
mxArray * prhs[],const char * command_name
);

函数功能:

在 MEX 文件中调用 Matlab 的函数或者用户编写的函数及 MEX 文件。

参数说明：

下面参数的定义与 mexFunction 的定义、构造和使用方法相同。

int nlhs　调用函数的输出参数个数。
mxArray * plhs[]　调用函数的输出参数。
int nrhs　调用函数的输入参数个数。
mxArray * prhs[],const char * command_name　调用函数的输入参数。

```
/ * mexcallMatlab.c * /
/ * 说明：
        实例调用 mexCallMatlab 调用用户编写的函数 image3d(image3d.m)
 * /
# include "mex.h"
void mexFunction( int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    mexCallMatlab(0,NULL,0,NULL,"image3d");
}
```

image3d 函数是将二维图像按照三维数据的形式显示出来(在本书附带的光盘中可以找到 image3d.m 文件)。实例的执行结果如图 3-3 所示。

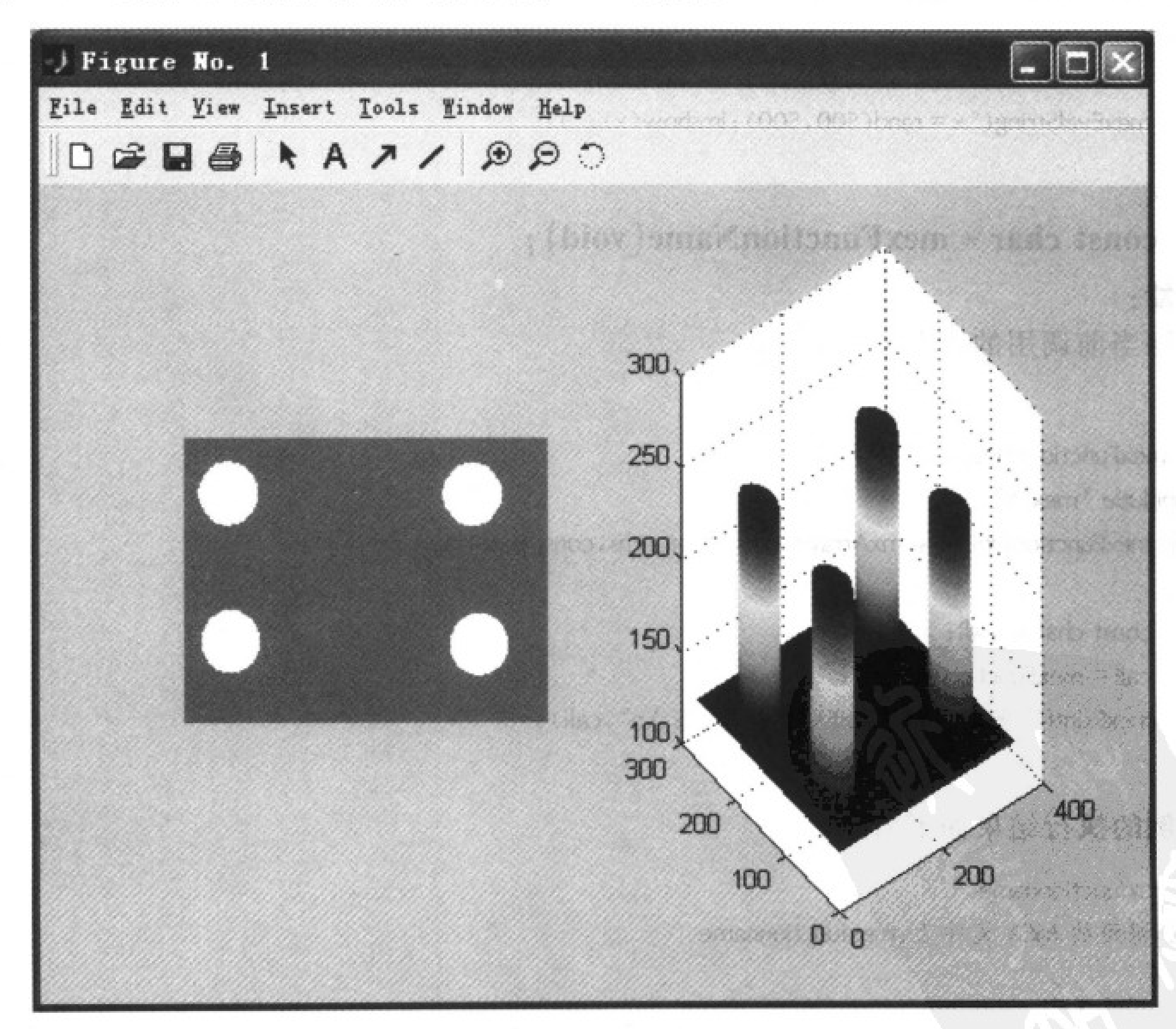


图 3-3　mexCallMatlab 实例运行结果

3. void mexErrMsgTxt(const char * error_msg);

函数功能:

输出错误信息。

4. int mexEvalString(const char * command);

函数功能:

执行 Matlab 命令。

参数说明:

const char * command　要执行的 Matlab 命令。

附加说明:

mexEvalString 与 mexCallMatlab 不同,mexCallMatlab 可以传递 MEX 文件中的参数并且将执行结果返回到 MEX 文件中,而 mexEvalString 则只能调用当前工作空间中的变量进行运算。

实例:

```
/* mexEvalString 文件内容 */
#include "mex.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    mexPrintf("x=rand(500,500);imshow(x);");
    mexEvalString("x=rand(500,500);imshow(x);");
}
```

5. const char * mexFunctionName(void);

函数功能:

返回当前调用的 MEX 文件名。

实例:

```
/* mexFunctionName.c 文件内容 */
#include "mex.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    const char * call;
    call=mexFunctionName();
    mexPrintf("当前调用的 MEX 文件为:%s\n",call);
}
```

实例的执行结果如下所示。

```
>> mexfunctionname
当前调用的 MEX 文件为:mexfunctionname
>>
```

6. const mxArray * mexGet(double handle,const char * property);

函数功能:

得到 Matlab 指定图形句柄的属性值。

参数说明：

double handle　Matlab 图形绘制窗口句柄。

const char * property　待返回值的属性名称。

附加说明：

实例：

```
/* mexget.c 文件内容 */
#include "mex.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,
                 const mxArray * prhs[])
{
    double              handle;
    const mxArray *     position;
    double *            pr;

    /* 参数有效性检查 */
    if(nrhs != 1 || ! mxIsDouble(prhs[0]))
    {
    mexErrMsgTxt("只接收一个参数,并且为 double 型 Matlab 句柄! \n");
    }
    position = mexGet( * mxGetPr(prhs[0]),"Position");
    pr = mxGetPr(position);
    pr[2] = pr[2]/2;/* 把输入的窗口句柄对应的窗口变为原来大小的一半 */
    pr[3] = pr[3]/2;
    mexPrintf("现在窗口的参数如下: \n");
    mexPrintf("right: %f,left: %f,width: %f,hegith: %f",pr[0],pr[1],pr[2],pr[3]);
    mexSet( * mxGetPr(prhs[0]),"Position",position);
}
```

实例的执行结果如下所示：

```
>> h = figure;mexget(h);
```

现在窗口的参数如下：

right:232.000000,left:258.000000,width:280.000000,hegith:210.000000

7. mxArray * mexGetVariable(const char * workspace,const char * var_name);

函数功能：

从指定工作区间中根据变量名称得到该变量的一个备份。

参数说明：

const char * workspace 工作区间,其中 'global' 表示全局工作空间,'base' 表示基本工作空间,'caller' 表示调用此 MEX 文件的工作空间。

const char * var_name　变量名称字符串。

函数返回：

如果调用成功则返回相应变量复制的 mxArray 指针;如果调用失败,则返回 NULL。

附加说明：

用此函数得到的只是变量的一个复制，对得到的 mxArray 指针进行任何修改，不会影响工作区间中的原变量。

8. const mxArray * mexGetVariablePtr(const char * var_name, const char * workspace);

函数功能：

根据变量名称，得到指定工作区间变量的只读 mxArray 指针。

参数说明：

const char * var_name　变量名称字符串。

const char * workspace　工作区间，其中 'global' 表示全局工作空间，'base' 表示基本工作空间，'caller' 表示调用此 MEX 文件的工作空间。

附加说明：

用 mexGetVariablePtr 得到的变量指针是只读的，不能修改变量的内容。如果需要改变变量的内容，可以采用 mexGetVariable 和 mexSetVariable 组合的方式。

9. bool mexIsGlobal(const mxArray * array_ptr);

函数功能：

判断输入变量是否为全局工作变量。

10. bool mexIsLocked(void);

函数功能：

判断当前 MEX 文件是否被锁定(锁定即不能被用户在命令行清除掉)，默认情况下 MEX 文件是非锁定的。

11. void mexLock(void);

函数功能：

锁定当前的 MEX 文件。

12. void mexMakeArrayPersistent(mxArray * array_ptr);

函数功能：

使输入的 Matlab mxArray 阵列在 MEX 文件退出以后仍然存在。

参数说明：

mxArray * array_ptr Matlab　阵列 mxArray 的指针。

附加说明：

一般情况下，使用 mxCreate * * 函数创建的 mxArray 变量在 MEX 文件退出时，Matlab 内存管理器会自动将其占用的内存释放。如果开发者希望创建的 mxArray 变量在当前 MEX 文件退出以后，仍然可以被 MEX 文件访问到，就要使用 mexMakeArray-Persistent 函数改变其生命周期。但是，Matlab 内存管理器并不负责释放 mxArray 变量占用的内存，因而需要结合 mexAtExit 函数使用，注册一个 MEX 文件被清除时需要调用的函数，并在此函数中调用 mxDestroyArray 来释放其内存。

13. void mexMakeMemoryPersistent(void * ptr);

函数功能：

改变由 Matlab 的内存分配函数(如 mxCalloc，mxMalloc，mxRealloc 等)分配的内存的生

命周期，使其在 MEX 文件执行完毕时，仍然保持在内存中。

附加说明：

与 mexMakeArrayPersistent 类似，用此函数改变了内存的生命周期以后，Matlab 内存管理器不会自动释放这块内存，因而需要用 mexAtExit 函数注册一个在系统退出或者 MEX 文件被清除时调用的函数以释放内存。

实例：

```
/ * mexmakepersist.c 文件内容 * /
/ * 说明：
    执行 mexmakepersist('c')创建长生命周期的变量
    执行 mexmakepersist('s')显示创建的长生命周期变量
    采用 clear all 清除 MEX 文件时，freememory 函数会自动清除长生命周期变量的内存空间
 * /
  # include <stdio.h>
  # include "mex.h"
  # include "stdlib.h"
  # include "string.h"

  / * 静态全局变量，MEX 文件执行退出时仍然存在！ * /
  mxArray *  pArray=NULL;
  char *  strBuff=NULL;
  int isMexAtExitCall=NULL;

  / * 当 MEX 文件被清除或者 Matlab 退出时调用此函数 * /
  static void freememory(void)
  {
       if(pArray!  =NULL)
         {
              mxDestroyArray(pArray);
        pArray=NULL;
        }
       if(strBuff!  =NULL)
        {
              mxFree(strBuff);
                 pArray=NULL;
          }
       mexPrintf("释放全局生命周期变量的内存！\n");
}

void mexFunction(int nlhs,mxArray *  plhs[],int nrhs,const mxArray * prhs[])
{
     int i=0;
     mxArray *  pString;
     char *  buff;
     mxArray *  prhs1[1];
```

```
pString = prhs[0];
if((nrhs! = 1)||!(mxIsChar(pString)))
    {
        mexErrMsgTxt("只接收一个字符类型的 Matlab 阵列! \n");
    }
buff = mxCalloc(2,sizeof(char));
mxGetString(pString,buff,2);
printf("输入的命令为:%c\n", * buff);
if( * buff == 'c')
    {
        if(pArray == NULL)
            {
                pArray = mxCreateDoubleMatrix(512,512,mxREAL);
                    for(i = 0;i<512 * 512;i++ )
                        {
                            * (mxGetPr(pArray) + i) = (1.0 * rand())/RAND_MAX;
                        }
                mexMakeArrayPersistent(pArray);
                printf("生成 512 * 512 随机矩阵! \n");
            }
        if(strBuff == NULL)
        {
            strBuff = mxCalloc( 100,sizeof(char));
            for(i = 0;i<99;i++ )
            {
                strBuff[i] = (char)('A' + (1.0 * rand()/RAND_MAX) * 25);
            }
            strBuff[i] = '\0';
            mexMakeMemoryPersistent(strBuff);
            printf("生成 99 个随机大写字母! \n");
            printf(" %s\n",strBuff);
        }
    }
 else if( * buff == 's')
    {
        if(strBuff ! = NULL)
        {
            mexPrintf("在第二个 mex 函数中显示 strBuff 的内容:\n");
            mexPrintf(" %s\n",strBuff);
        }

        if(pArray! = NULL)
        {
            mexEvalString("pause(3)");
            prhs1[0] = pArray;
```

```
                mexPrintf("显示 mexmakepersist 生成的随机矩阵！\n");
                mexCallMatlab(0,NULL,1,prhs1,"imshow");
            }
        }
    else
        {
            mexPrintf("输入命令标志错误！\n");
        }
    if(isMexAtExitCall == 0)
        {
            mexAtExit(freememory);
            isMexAtExitCall = 1;
            mexPrintf("MEX 文件被清除时的退出函数已经注册！\n");
        }
    mxFree(buff);
    buff = NULL;
}
```

实例的执行结果如图 3－4 所示。

```
>> mex mexmakepersist.c
>> mexmakepersist('c');
```

输入的命令为：c

```
生成 512 * 512 随机矩阵！
生成 99 个随机大写字母！
OFBCFYQNXNDYQTYHDEOCYPBPVCSAWDMPWFTHRJUBASWOJOEMATRA
```

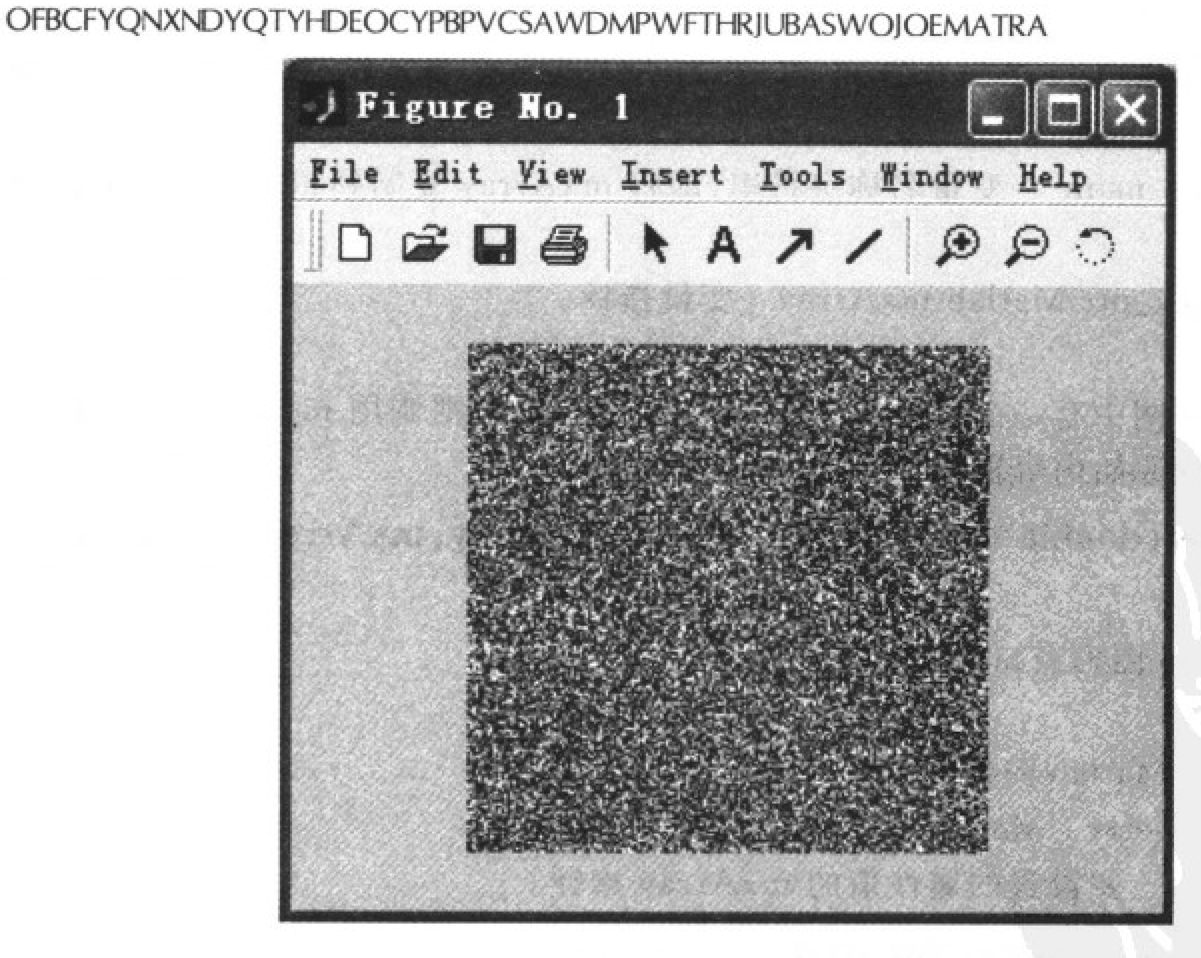


图 3－4　mexmakepersist 产生的随机矩阵显示

```
MVOFXCATSSVBGAIPAREGJEVTBCDMGXHSCYNIXHBFKCWDMLJ
MEX 文件被清除时的退出函数已经注册!
>> mexmakepersist('s');

输入的命令为:s

在第二个 mex 函数中显示 strBuff 的内容:
OFBCFYQNXNDYQTYHDEOCYPBPVCSAWDMPWFTHRJUBASWOJOEMATRA
MVOFXCATSSVBGAIPAREGJEVTBCDMGXHSCYNIXHBFKCWDMLJ
显示 mexmakepersist 生成的随机矩阵!
>>
>> clear all;
释放全局生命周期变量的内存!
```

14. int mexPrintf(const char * format,...);

函数说明:

使用方法与 C 语言的 printf 函数相同。

附加说明:

由于 Matlab 已经内置了一个 printf 函数,因而采用 mexPrintf 函数可以使 MEX 文件不必链接整个 stdio 链接库。

15. int mexPutVariable(const char * workspace,const char * var_name, mxArray * array_ptr);

函数说明:

将指定 Matlab mxArray 变量复制到相应的工作空间中。

参数说明:

const char * workspace　工作区间,其中 'global' 表示全局工作空间,'base' 表示基本工作空间,'caller' 表示调用此 MEX 文件的工作空间。

const char * var_name　变量名称字符串,即将 mxArray 复制到相应工作空间的变量名称。

mxArray * array_ptr Matlab mxArray　变量指针。

附加说明:

如果目的工作空间中有一个变量名称与 var_name 相同,则调用 mexPutVariable 会将目的工作空间中变量的原始内容覆盖。

16. int mexSet(double handle,const char * property,mxArray * value);

函数功能:

设置相应 Matlab 图形窗口的属性值。

参数说明:

double handle　Matlab 图形句柄。

const char * property　Matlab 图形句柄的属性名称。

mxArray * value　要设置的属性值的 mxArray 指针。

17. void mexSetTrapFlag(int trap_flag);

函数功能:

设置调用 mexCallMatlab 出错时的处理策略。

参数说明：

int trap_flag 处理策略标志，为 0 表示在调用 mexCallMatlab 函数出错时返回到 Matlab 命令行中，为 1 表示在调用 mexCallMatlab 函数出错时返回到 MEX 文件中。默认情况下，在调用 mexCallMatlab 函数出错时返回到 Matlab 命令行中。

18. void mexUnlock(void);

函数功能：

结束当前 MEX 文件的锁定状态，使用户可以通过命令行从内存中清除 MEX 文件。

19. void mexWarnMsgTxt(const char * warning_msg);

函数功能：

显示警告信息，与 mexErrMsgTxt 函数的使用方法类似。

3.8 Matlab 普通数值阵列的操作

Matlab 双精度型数值阵列的实部数据区的指针和虚部数据区的指针分别可以通过 mxGetPr 和 mxGetPi 这两个函数得到，因而 C 语言对 Matlab 双精度型数值阵列的操作只要通过操作其实部数据区指针和虚部数据区指针即可。

Matlab 其他类型的数值阵列类型如 8 位整型、16 位整型和 32 位整型等，其实部数据区的指针和虚部数据区的指针则是通过 mxGetData 和 mxGetImagData 这两个函数来得到，C 语言对这些类型的数值阵列的操作同样通过操作其实部数据区指针和虚部数据区指针。

需要注意的是，Matlab 数值阵列与 C 语言数组在内存中的存储方式是不一样的。Matlab 数值阵列在内存中的存储方式为按列的方式存储，而 C 语言数组在内存中的存储方式则是按行的方式存储。因而，当 Matlab 数值阵列大于 1 维的话，如果此时采用 mxGetPr、mxGetPi、mxGetData 和 mxGetImagData 得到的指针来操作其实部数据和虚部数据，那么，采用 int mxCalcSingleSubscript 计算每个元素相对于第一个元素存储的线性偏移量。

举例如下所示。

```
/* showmatrix.c 文件内容 */
#include "mex.h"
#include "matrix.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    int i,j,k;
    mwIndex index;
    double * pr=NULL;
    double * pi=NULL;
    size_t M,N;
    size_t ndim;
    mwSize dims[2];
    for(i=0;i<nrhs;i++)
    {
if((mxIsDouble(prhs[i]))&&(mxGetNumberOfDimensions(prhs[i])==2))
```

```
        {
            pr = mxGetPr(prhs[i]);
            pi = mxGetPi(prhs[i]);
            M = mxGetM(prhs[i]);
            N = mxGetN(prhs[i]);
            ndim = mxGetNumberOfDimensions(prhs[i]);

            mexPrintf("变量%d:\n",i);
            for(j = 0;j<M;j++)
            {
                dims[0] = j;
                for(k = 0;k<N;k++)
                {
                    dims[1] = k;
                    index = mxCalcSingleSubscript(prhs[i],ndim,dims);
                    if(pi == NULL)
                    {
                        mexPrintf("%6.2f,",pr[index]);
                        }
                    else
                    {
                        mexPrintf("%6.2f+%6.2fj,",pr[index],pi[index]);
                    }
                }
                mexPrintf("\n");
            }
        }
        else
        {
            mexPrintf("输入%d个变量不是二维 double 数值阵列!\n",i);
        }
    }
}
```

实例的实行结果如下所示。

```
>> a = rand(3,3);b = rand(4,4);c = rand(5,5) + j * rand(5,5);showmatrix(a,b,c);
变量 0:
  0.21,  0.13,  0.63,
  0.84,  0.21,  0.37,
  0.63,  0.61,  0.58,
变量 1:
  0.45,  0.01,  0.04,  0.02,
  0.04,  0.38,  0.61,  0.19,
  0.03,  0.68,  0.61,  0.59,
  0.31,  0.09,  0.02,  0.06,
```

```
)
变量 2:
  0.37+  0.24j,  0.45+  0.84j,  0.70+  0.34j,  0.45+  0.12j,  0.87+  0.26j,
  0.63+  0.05j,  0.44+  0.17j,  0.73+  0.31j,  0.72+  0.04j,  0.23+  0.16j,
  0.72+  0.08j,  0.35+  0.17j,  0.48+  0.37j,  0.89+  0.46j,  0.80+  0.87j,
  0.69+  0.64j,  0.15+  0.99j,  0.55+  0.39j,  0.27+  0.87j,  0.91+  0.24j,
  0.08+  0.19j,  0.68+  0.44j,  0.12+  0.59j,  0.25+  0.93j,  0.23+  0.65j,
>>
```

3.9 稀疏数组阵列(Sparse Array)

Matlab 的 Sparse 数组与一般的数组存储方式不一样，采用 Sparse 数组可以使零元素个数很多的数组占用较少的内存空间。在 Matlab 与 C 语言混合编程时，Sparse 数组用参数 pr、pi、ir、jc 和 nzmax 来描述。其中各参数的意义如下。

1. pr

double * pr　依次存放 Sparse 数组中非零元素的实部值。

2. pi

double * pi　依次存放 Sparse 数组中非零元素的虚部值。如果数组的所有元素为实数，则 pi 为空。

3. ir

mwIndex * ir　依次存放 Sparse 数组中非零元素的行值。

4. jc

mwIndex * jc　存放 Sparse 数组中非零元素的列信息。其数组元素个数为 N+1(N 为 Sparse 数组的列数)。所有非零元素在 pr、pi 和 ir 中的存放顺序同样是按列存放的。0≤j≤N－1 时，jc[j]代表 Sparse 数组第 j 列的非零元素在 pr、pi、ir 中的第一个索引值，而 jc[j+1]－1 则代表第 j 列的非零元素在 pr、pi 和 ir 中的最后一个索引值。

0	1	0	0	0
0	2	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

图 3-5　矩阵元素分布图

假定 5×5 实数矩阵，其矩阵元素的分布如图 3-5 所示。

如果 Sparse 的 nzmax＝12，则 pr、ir 和 jc 的分布如图 3-6 所示。

参照图 3-5 和图 3-6 给出从 jc、ir 和 pr 求出第 2 列的所有非零元素。由于 jc[1]＝0，jc[2]＝2，则 ir[jc[1]:jc[2]－1]＝ir[0:1]为第 2 列的所有非零元素，即第 2 列的非零元素为 Sparse 矩阵的(1,2)，(2,2)，其值分别为 1 和 2。这样，用 Sparse 矩阵表示方式就可以将一个零元素个数很多的矩阵用一种节省内存的方式保存起来。

5. nzmax

mwSize nzmax　Sparse 中可以存放的非零元素的最大数目，其中 pr，pi 和 ir 的数组大小均为 nzmax。

下面是一个显示稀疏矩阵内容的实例。

```
/* mxshowsparsematrix.c 文件内容 */
```

pr	ir	jc	索引
1	0	0	0
2	1	0	1
1	2	2	2
		3	3
		3	4
		3	5
		*	6
		*	7
		*	8
		*	9
		*	10
		*	11

注：
(1) *表示没有元素；
(2) 灰色的元素格表示有元素，但是其值对当前Sparse矩阵无意义；
(3)所有C语言的数组从0开始计数。

图 3-6 Sparse 各参数的元素分布图

```
#include "mex.h"
#include "matrix.h"

void showSparseMatrix(mxArray * pa)
{
    if(pa == NULL)
    {
        return;
    }
    if(mxIsSparse(pa))
    {
        mwIndex * ir, * jc;
        double * pr, * pi;
        size_t M,N;
        int i,j;
        mwSize curColumNZnum = 0;/ * 每列的非零元素个数 * /
        mwSize nznum = 0;/ * 已知的非零元素个数 * /
        mwSize nzmax = mxGetNzmax(pa);
        N = mxGetN(pa);
        M = mxGetM(pa);
        ir = mxGetIr(pa);
        jc = mxGetJc(pa);
        pr = mxGetPr(pa);
        pi = mxGetPi(pa);
        for(i = 0;i<N;i++)
        {
            curColumNZnum = jc[i+1] - jc[i];
            while(curColumNZnum>0)
            {
                if(pi == NULL)
```



```
                {
                    mexPrintf("\t<%d,%d> = %6.2f\n",
                              ir[nznum]+1,i+1,pr[nznum]);
                }
                else
                {
                    mexPrintf("\t<%d,%d> = %6.2f + %6.2fi\n",
                              ir[nznum]+1,i+1,pr[nznum],pi[nznum]);
                }
                curColumNZnum--;
                nznum++;
            }
        }
    }
    else
    {
        mexPrintf("\t输入变量非稀疏矩阵! \n");
    }
}

void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    int i=0;
    for(i=0;i<nrhs;i++)
    {
        mexPrintf("第%d个输入参数:\n",i+1);
        showSparseMatrix(prhs[i]);
    }
}
```

用 mex mxshowsparsematrix.c 编译上述 MEX 文件，并采用下面的 m Script 文件测试 mxshowsparsematrix 的功能。

```
>> v=sign((rand(5,5)-0.8)); %产生一个随机矩阵，并将>0.8的元素置为1
>>v1=(v+abs(v))/2;
>>[i,j,vnz]=find(v1);%寻找v1中>0的元素
>>v2=sparse(i,j,vnz);%构造稀疏矩阵v2
>>disp(v2);mxshowsparsematrix(v2,v1);%对比disp函数和mxshowsparsematrix的功能
```

测试显示的结果如下所示。

```
   (3,1)        1
   (1,2)        1
   (2,4)        1
   (3,5)        1
   (4,5)        1
第1个输入参数:
```

```
    <2,2> =   1.00
    <4,2> =   1.00
    <1,3> =   1.00
    <2,3> =   1.00
    <4,5> =   1.00
第 2 个输入参数：
    输入变量非稀疏矩阵！
>>
```

3.10 Matlab 元组

Matlab 元组阵列元素的内容是一个 Matlab 阵列，对 Matlab 元组阵列来说，其主要操作是用 mxGetCell 取得 Matlab 元组阵列元素的内容，用 mxSetCell 设置 Matlab 元组元素的内容。下面的实例演示了如何通过调用 mxGetCell 函数得到 Matlab 元组阵列的内容并显示其类型。

```
/* showcell.c 文件内容 */
#include "mex.h"
#include "matrix.h"

void printArrayClass(mxArray * pa)
{
    if(pa == NULL)
    {
        mexPrintf("空");
        return;
    }
    switch(mxGetClassID(pa))
    {
            case mxCELL_CLASS:
            {
                mexPrintf("元组阵列");
                break;
            }
        case mxSTRUCT_CLASS:
            {
                mexPrintf("结构体阵列");
                break;
            }
        case mxCHAR_CLASS:
            {
                mexPrintf("字符串阵列");
                break;
            }
        case mxLOGICAL_CLASS:
```



```
            {
                mexPrintf("逻辑数值阵列");
                break;
            }
        case mxDOUBLE_CLASS:
            {
                mexPrintf("双精度数值阵列");
                break;
            }
        case mxSINGLE_CLASS:
                {
                    mexPrintf("单精度数值阵列");
                    break;
            }
        case mxINT8_CLASS:
        case mxINT16_CLASS:
        case mxINT32_CLASS:
        case mxINT64_CLASS:
                {
                    mexPrintf("有符号整型数值阵列");
                    break;
            }

        case mxUINT64_CLASS:
        case mxUINT8_CLASS:
        case mxUINT16_CLASS:
        case mxUINT32_CLASS:
                {
                    mexPrintf("无符号整型数值阵列");
                    break;
            }
        default:
                {
                    mexPrintf("其他 MATLAB 类型阵列");
                    break;
            }
    }
}

void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    mxArray *  pa=NULL;
    mwSize cellNum=0;
    int i,j;
    for(i=0;i<nrhs;i++)
```

```
    {
        pa = prhs[i];
        if(mxIsCell(pa))
        {
            cellNum = mxGetNumberOfElements(pa);
            mexPrintf("第%d个输入变量的各元素类型如下:\n",i+1);
            for(j=0;j<cellNum;j++)
            {
                mexPrintf("\t第%d个元素是:",j);
                printArrayClass(mxGetCell(pa,j));
                mexPrintf("\n");
            }
        }
        else
        {
            mexPrintf("第%d个输入变量非元组阵列! \n",i+1);
        }
    }
}
```

3.11 Matlab 结构体阵列

结构体阵列实际上一个结构体的数组，数组的每个元素是相同结构的结构体，如表 3-6 所列。如图 3-7 所示，每个结构体元素由结构体域构成，其中通过域号(filed number)或者域名(field name)来区分每一个结构体域。结构体域名由开发者指定，结构体的域号根据各域的存储顺序自动分配，第一域的域号为 0，最后一个域的域号为 M-1，其中 M 为结构体元素总的域的个数。结构体各域的内容可以为任意 Matlab 阵列类型。下面的实例演示了如何遍历 Matlab 结构体阵列并显示其类型。

表 3-6　Matlab 结构体阵列存储示意图

索　引	1	2	…	N
内　容	STRUCT	STRUCT	…	STRUCT

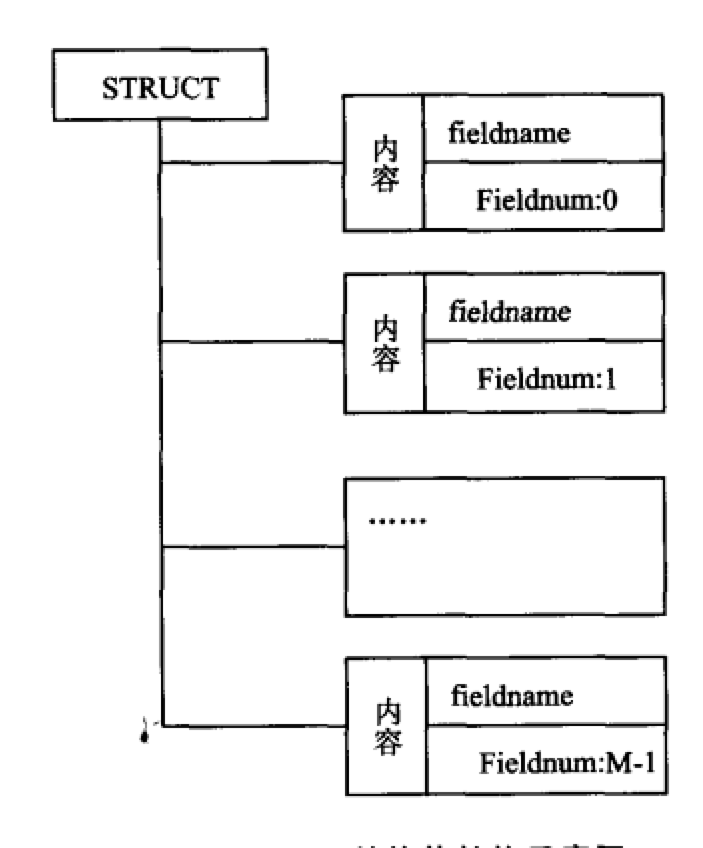


图 3-7　Matlab 结构体结构示意图

```
/* showstruct.c 文件内容 */
#include "mex.h"
#include "matrix.h"
#define _MAX_STRUCT_FIELD_NUM 40
```

```
void printArrayClass(mxArray * pa)
{
    if(pa == NULL)
        {
            mexPrintf("空");
            return;
        }
    switch(mxGetClassID(pa))
        {
                    case mxCELL_CLASS:
                    {
                        mexPrintf("元组阵列");
                        break;
                        }
                case mxSTRUCT_CLASS:
                    {
                        mexPrintf("结构体阵列");
                        break;
                    }
                case mxCHAR_CLASS:
                    {
                        mexPrintf("字符串阵列");
                        break;
                    }
                case mxLOGICAL_CLASS:
                    {
                        mexPrintf("逻辑数值阵列");
                        break;
                    }
                case mxDOUBLE_CLASS:
                    {
                        mexPrintf("双精度数值阵列");
                        break;
                    }
                case mxSINGLE_CLASS:
                        {
                            mexPrintf("单精度数值阵列");
                            break;
                    }
                case mxINT8_CLASS:
                case mxINT16_CLASS:
                case mxINT32_CLASS:
                case mxINT64_CLASS:
```

```
                {
                    mexPrintf("有符号整型数值阵列");
                    break;
                }

            case mxUINT64_CLASS:
            case mxUINT8_CLASS:
            case mxUINT16_CLASS:
            case mxUINT32_CLASS:
                {
                    mexPrintf("无符号整型数值阵列");
                    break;
                }
            default:
                {
                    mexPrintf("其他 MATLAB 类型阵列");
                    break;
                }
        }
}

void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    mxArray *  pa=NULL;
    int i,j;
    char *  fieldname=NULL;
    for(i=0;i<nrhs;i++)
    {
        if(mxIsStruct(prhs[i]))
        {
            mexPrintf("第%d个输入变量的各域类型如下:\n",i+1);
            for(j=0;j<_MAX_STRUCT_FIELD_NUM;j++)
                {
                    fieldname=mxGetFieldNameByNumber(prhs[i],j);
                    if(fieldname==NULL)
                        {
                            break;
                        }
                    mexPrintf("\t%s:",fieldname);
                    printArrayClass(mxGetFieldByNumber(prhs[i],0,j));
                    mexPrintf("\n");
                }
        }
```

```
            else
            {
                mexPrintf("第%d个输入变量非结构体阵列！\n",i+1);
            }
        }
    }
```

实例执行结果如下所示。

```
>> a=rand(1,1);
   b='哈尔滨工程大学';
   c=struct('name1',rand(3),'name2','nothing');
   d={a,b,c};
   e=struct('rand1',a,'university',b,'struct',c,'cell',d);
   showstruct(e,a);
第 1 个输入变量的各域类型如下：
   rand1:双精度数值阵列
   university:字符串阵列
   struct:结构体阵列
   cell:双精度数值阵列
第 2 个输入变量非结构体阵列！
>>
```

3.12 Matlab 字符阵列

Matlab 字符阵列的的基本元素是类型 mxChar，Matlab 的字符类型的 mxArray 都是用 mxChar 的字符类型，而不是 C 语言的 char 型。Matlab API 将 mxChar 定义为 2 字节的无符号整型，这样可以方便的处理 UNICODE 字符。因而在用 C 语言编写 MEX 文件源代码时，如果要从 Matlab 字符类型的 mxArray 中导出字符串的话，需要注意的是 mxArray 使用的是 mxChar 类型的字符，因而需要分配比 char 多一倍的内存空间。如下面的例子 mxshowstring. c：

```
/* mxshowstring.c 文件内容 */
#include "mex.h"
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
    char *buf;
    size_t   buflen;
    int i=0;

    if (! mxIsChar(prhs[0]) || (mxGetM(prhs[0]) != 1))
    {
        mexErrMsgTxt("输入数组必须是一维字符串");
```

```
    }

    for(i=0;i<nrhs;i++)
    {
        if(mxIsChar(prhs[i]))
        {
            buflen=mxGetNumberOfElements(prhs[i]) * sizeof(char) + 1;
            buf=mxMalloc(buflen);
            mxGetString(prhs[i],buf,buflen);
            mexPrintf("输入%d 个字符阵列为：  %s\n",i,buf);
            mxFree(buf);
        }
        else
        {
            mexPrintf("第%d 个输入阵列不是字符类型！\n");
        }
    }
}
```

实例执行结果如下所示。

```
>> mxshowstring('string1','string2','string3',rand(3,3),'string4');
输入 0 个字符阵列为：  string1
输入 1 个字符阵列为：  string2
输入 2 个字符阵列为：  string3
第 5 个输入阵列不是字符类型！
输入 4 个字符阵列为：  string4
>>
```

3.13 Matlab mat API 函数

MAT 格式的文件是 Matlab 的文件格式，如果外部数据与 Matlab 进行复杂格式的数据交换的话，MAT 文件是最好的选择。Matlab 提供了一系列进行 MAT 操作的 API 函数，这些 API 函数都以 mat 开头，被称为 mat 函数。现将其中的函数的使用方法一一介绍如下所述。

1. int matClose(MATFile * mfp);

函数功能：

关闭 MAT 文件。

参数说明：

MATFile * mfp　要关闭的 MAT 格式文件指针。

2. int matDeleteVariable(MATFile * mfp,const char * name);

函数功能：

删除 MAT 文件中名称为 name 的变量。

参数说明：

MATFile * mfp　MAT 文件指针。

const char * name　要删除的变量名称。

3. char * * matGetDir(MATFile * mfp, int * num);

函数功能：

得到 MAT 文件的变量列表。

参数说明：

MATFile * mfp　MAT 文件指针。

int * num　整型变量指针，用以返回 MAT 文件中的变量个数。

附加说明：

matGetDir 返回一个 MAT 文件中的 Matlab 变量列表，其中每个变量名称的最大长度为 mxMAXNAM（在 matrix.h 中定义，其值默认为 64）。如果函数调用失败，此时 *num = -1，并且返回 NULL 值。如果 MAT 文件中不包含任何 Matlab 变量，*num = 0。下面的实例就是利用 matGetDir 函数来显示输入 MAT 文件中存储的变量名称。

```
/* matGetDir.c 文件内容 */
#include "mex.h"
#include "mat.h"

int listvariable(const char * file)
{
MATFile * pmat;
const char * * dir;
const char * name;
int  ndir;
int  i;

/* 打开 MAT 文件 */
pmat = matOpen(file, "r");
if (pmat == NULL)
    {
        mexPrintf("打开文件：%s 出错！\n", file);
        return(1);
    }

/*   得到 MAT 变量的目录列表   */
dir = (const char * * )matGetDir(pmat, &ndir);
if (dir == NULL)
    {
        mexPrintf("读取文件 %s 的变量列表出错！\n", file);
        return(1);
```

```
        }
    else
        {
            mexPrintf("MAT 文件 %s 中的变量如下:\n",file);
            for (i=0; i < ndir; i++ )
                {
                    mexPrintf(" %s\n",dir[i]);
                }
    }
    mxFree(dir);

    /* 关闭 MAT 文件. */
    if (matClose(pmat) ! = 0)
    {
        mexPrintf("关闭文件 %s 出错! \n",file);
      return(1);
    }
    return(0);
}

void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
        int i=0;
        char *  buff=NULL;
        buff=mxCalloc(200,sizeof(char));
        for(i=0;i<nrhs;i++ )
        {
          if(mxIsChar(prhs[0]))
                {
                    mxGetString(prhs[i],buff,200);
                    listvariable(buff);
                }
          else
                {
                    mexPrintf("输入的第%d个变量不是字符类型阵列! \n");
                }
        }
mxFree(buff);
}
```

该 MEX 文件的测试程序如下所示。

```
>>clear all
>>mex matGetDir.c
```

```
>>a = rand(1,1);b = magic(5);c = 'nothing';
>>save 'abc.mat';matGetDir('abc.mat')
```

MAT 文件 abc.mat 中的变量如下：

```
a
c
b
>>
```

4. FILE * matGetFp(MATFile * mfp);

函数功能：

得到 MAT 文件的 C 语言类型文件指针(FILE *)。

5. mxArray * matGetNextVariable(MATFile * mfp,const char * * name);

函数功能：

得到 MAT 文件的下一个 Matlab 阵列变量。

参数说明：

MATFile * mfp　MAT 文件指针。

const char * * name　用于返回 MAT 文件中文件名的字符串指针。

附加说明：

此函数必须紧接着 matOpen 函数调用成功之后使用，如果在 matOpen 函数和 matGetNextVariable函数之间还有另外的 mat-函数的调用，否则返回的值是不可预测的。另外，使用此函数时，需要用 mxDestroyArray 释放得到 Matlab 变量 mxArray 指针所指向的内存空间。

下面的实例就是用 matGetNextVariable 函数来显示输入 MAT 文件中变量的维数信息。

```
/* matGetNextVariable.c 文件内容 */
#include "mex.h"
#include "mat.h"

int listvariabledims(const char * file)
{
    MATFile * pmat;
    const char * name = NULL;
    int         i;
    mxArray * pa = NULL;
    mwSize ndim;
    mwSize * dim;

    /* 打开 MAT 文件 */
    pmat = matOpen(file,"r");
    if (pmat == NULL)
        {
```

```
            mexPrintf("打开文件:%s 出错! \n",file);
            return(1);
        }

    mexPrintf("MAT 文件 %s中的变量及信息如下:\n",file);
    /* 得到 MAT 文件中的各个变量 */
    pa = matGetNextVariable(pmat,&name);
    while(pa! =NULL)
        {

            mexPrintf("变量%s   ",name);
            dim = mxGetDimensions(pa);
            ndim = mxGetNumberOfDimensions(pa);
            mexPrintf("维数信息:");
            for(i = 0;i<ndim - 1;i++)
                {
                    mexPrintf(" %dx",dim[i]);
                }
            mexPrintf(" %d\n",dim[i]);
            mxDestroyArray(pa);
            pa = matGetNextVariable(pmat,&name);
        }
        /* 关闭 MAT 文件. */
        if (matClose(pmat) ! = 0)
            {
                mexPrintf("关闭文件 %s 出错! \n",file);
                return(1);
            }
    return(0);
}
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
    int i = 0;
    char *  buff = NULL;
    buff = mxCalloc(200,sizeof(char));
    for(i = 0;i<nrhs;i++)
        {
            if(mxIsChar(prhs[0]))
                {
                    mxGetString(prhs[i],buff,200);
                    listvariabledims(buff);
                }
            else
```

```
                {
                    mexPrintf("输入的第%d个变量不是字符类型阵列！\n");
                }
            }
    mxFree(buff);
}
```

测试程序及输出结果如下所示。

```
>> a=rand(1,1);b=magic(5);c='nothing';d={a,b,c};
>> save abcd.mat;matgetnextvariable('abcd.mat');
```

MAT 文件 abcd.mat 中的变量及信息如下：

```
变量 a   维数信息：1x1
变量 d   维数信息：1x3
变量 c   维数信息：1x7
变量 b   维数信息：5x5
>>
```

6. mxArray * matGetNextVariableInfo(MATFile * mfp,const char * * name);

函数功能：

得到 MAT 文件中变量的 mxArray 指针，但此指针中只包含 Matlab 阵列变量的头信息。

参数说明：

MATFile * mfp　MAT 文件指针。

const char * * name　用于返回 MAT 文件中文件名的字符串指针。

附加说明：

返回 Matlab 阵列的 mxArray 变量的 pr、pi、ir 和 jc 域均为空。

7. mxArray * matGetVariable(MATFile * mfp,const char * name);

函数功能：

从 MAT 文件读取指定名字的变量。

参数说明：

MATFile * mfp　MAT 文件指针。

const char * name　变量名称。

附加说明：

注意释放返回的 mxArray 指针所指向的内存。

8. mxArray * matGetVariableInfo(MATFile * mfp,const char * name);

函数功能：

从 MAT 文件读取指定名字的变量，只是此函数返回 mxArray 变量只变量的头信息，即 mxArray 变量的 pr、pi、ir 和 jc 域均为 NULL。

9. MATFile * matOpen(const char * filename,const char * mode);

函数功能：

打开一个 MAT 文件。

参数说明:

const char * filename　MAT 文件名;

const char * mode　打开 MAT 文件的选项。

附加说明:

matOpen 可以采用不同的选项,如表 3-7 所列。

表 3-7　matOpen 函数的选项及其含义

选　项	含　义
r	以只读方式打开 MAT 文件
u	以更新文件的方式打开 MAT 文件,可以读写文件。如果要打开的 MAT 文件不存在,不创建新的文件
w	以只写的方式打开文件,如果存在一个与待待开的文件的文件名相同的文件,则覆盖元文件并删除原文件的内容
w4	创建一个与 Matlab 4.0 及其以前所有的 Matlab 版本兼容的 MAT 文件
wL	采用此选项创建的 Matlab MAT 文件采用本机的默认的字符集,而不是默认情况下采用 UNICODE 字符集的 MAT 文件。采用此选项创建的 MAT 文件只能被 Matlab 6.5 以后的 Matlab 读取
wz	采用此选项创建的 Matlab MAT 文件采用压缩格式存储数据

10. int matPutVariable(MATFile * mfp,const char * name,const mxArray * mp);

函数功能:

向 MAT 文件中写入变量。

参数说明:

MATFile * mfp　MAT 文件指针。

const char * name　要写入的变量的名称。

const mxArray * mp　要写入的变量的 mxArray 指针。

参数返回值:

函数调用成功返回 0,否则返回非零值。

11. int matPutVariableAsGlobal(MATFile * mfp,const char * name,const mxArray * mp);

函数功能:

向 MAT 文件写入变量,并使此变量具备全局工作空间(global workspace)变量特性。

参数说明:

MATFile * mfp　MAT 文件指针。

const char * name　要写入的变量的名称。

const mxArray * mp　要写入的变量的 mxArray 指针。

参数返回值:

函数调用成功返回 0,否则返回非零值。

3.14 Matlab API 函数操作的实例

3.14.1 更改 Matlab 数值阵列的维数

要改变 mxArray 数组的各维数的大小，对于普通的数组而言，直接改变相应 mxArray 的维数信息数组即可。对于稀疏数组，情况就比较复杂了，需要重新计算并改变相应 mxArray 的 ir 和 jc 数组，下面给出的例子 mxsetdimensions.c 只针对二维稀疏矩阵进行处理，其源代码如下所示。

```
/* mxsetdimensions.c 文件内容 */
#include <string.h>
#include "mex.h"

void mexFunction(int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{

    mwSize number_new_dims,number_input_elements,number_new_elements,i;
    mwSize * new_dims;

    /* 检查输入参数的个数 */
    if (nrhs < 3)
    {
        mexErrMsgTxt("至少需要三个输入参数\n");
    }
    if(nlhs > 1){
        mexErrMsgTxt("输出参数太多\n");
    }
    number_new_dims = nrhs - 1;
    if (mxIsSparse(prhs[0]) && number_new_dims != 2)
    {
        mexErrMsgTxt("不支持多维稀疏矩阵\n");
    }

    number_input_elements = mxGetNumberOfElements(prhs[0]);

    /* 创建新的维数信息数组 */
    new_dims = mxMalloc(number_new_dims * sizeof( * new_dims));

    number_new_elements = 1;
    for (i=0; i< number_new_dims;i++)
    {
            const mxArray * pa;
```

```
            pa=prhs[i+1];
            if(mxGetNumberOfElements(pa) != 1)
            {
                mxFree(new_dims);
                mexErrMsgTxt("维数大小必须为 1*1 标量\n");
            }
            new_dims[i]=(int)mxGetScalar(pa);
            number_new_elements=new_dims[i]*number_new_elements;
    }
    if (number_new_elements != number_input_elements)
    {
        mxFree(new_dims);
        mexErrMsgTxt("改变后的数组个数必须和输入数组的个数相同.\n");
    }

    plhs[0]=mxDuplicateArray(prhs[0]);

    /* 针对 Sparse 数组和普通数组分不同情况进行处理 */
    if (mxIsSparse(plhs[0]))
    {
            size_t mold; /* 原来的行数 */
            size_t nold; /* 原来的列数 */
            mwIndex *jcold; /* 原来的 jc 数组 */
            mwIndex *ir; /* ir 数组 */
            size_t mnew; /* 新的行数 */
            size_t nnew; /* 新的列数 */
            mwIndex *jcnew; /* 新的 jc 数组 */
            int j,offset,offset1;

            jcnew=((int *)mxCalloc(new_dims[1]+1,sizeof(int)));
            mnew=new_dims[0];
            nnew=new_dims[1];
            mold=mxGetM(plhs[0]);
            nold=mxGetN(plhs[0]);
            jcold=mxGetJc(plhs[0]);
            ir=mxGetIr(plhs[0]);

                /************************************/
            /* 将输入数组看做一维线性数组,并改变 ir */
            for (i=1,offset=mold; i < nold; i++,offset+=mold)
            {
                for (j=jcold[i]; j < jcold[i+1]; j++)
                {
```

```
                    ir[j] += offset;
                }
            }

            /* 根据新的行数和列数信息,重新改变 ir 和设置 jcnew */
            for (i=0,j=0,offset=mnew-1,offset1=0; i < jcold[nold]; )
            {
                if (ir[i] > offset)
                {
                    jcnew[++j]=i;
                    offset   += mnew;
                    offset1 += mnew;
                }
                else
                {
                    ir[i++] -= offset1;
                }
            }

            /* 填充剩余的 jcnew 数组元素为 Sparse 数组非零元素的个数 */
            for (j++; j <= nnew; j++)
            {
                jcnew[j]=jcold[nold];
            }

            mxFree(mxGetJc(plhs[0]));
            mxSetJc(plhs[0],jcnew);
            mxSetM(plhs[0],mnew);
            mxSetN(plhs[0],nnew);
    }
    else
    {
            mxSetDimensions(plhs[0],new_dims,number_new_dims);
    }
    mxFree(new_dims);
}
```

为了说明 mxArray 为稀疏数组时候的维数大小改变的操作过程,下面以一个 6×6 的稀疏矩阵的维数大小由 6×6 改变为 4×9 的实例来说明其过程。此 6×6 稀疏矩阵的非零元素分布如图 3-8(a)所示。

而 ir 和 jc 在操作过程中的变化如图 3-8(b)所示,其中 irold 和 jcold 分别为输入稀疏矩阵的 ir 和 jc,ir 为将输入稀疏矩阵看做一维线性数组的时候的 ir 的值,irnew 和 jcnew 为根据输入参数改变以后的 ir 和 jc。上述操作过程的 Matlab 测试代码为:

```
sa = zeros(6,6);
sa(3,1) = 1;sa(1,2) = 1;sa(6,2) = 1;sa(5,6) = 1;
s = sparse(sa);
newsa = mxsetdimensions(s,4,9);
```

	1				
1					
					1
	1				

(a) 6×6稀疏矩阵的非零元素分布

irold	jcold	pr	ir	irnew	jcnew	索引
2	0	1	2	2	0	0
0	1	1	6	2	1	1
5	3	1	11	3	2	2
4	3	1	34	2	3	3
*	3	*	*	*	3	4
*	3	*	*	*	3	5
*	4	*	*	*	3	6
*	*	*	*	*	3	7
*	*	*	*	*	3	8
*	*	*	*	*	4	9

(b) ir和jc在操作过程中的变化

图 3-8　mxsetdimensions 对稀疏矩阵的操作过程

3.14.2　分析并显示 Matlab 阵列的内容

通过 Matlab 提供的以“mx”开头的函数及 Matlab 阵列的 mxArray * 类型指针,可以得到 Matlab 阵列的全部信息。下面的实例介绍了用“mx”开头的函数分析 Matlab 元组阵列(函数 analyze_cell)、结构体阵列(函数 analyze_structure)、字符串阵列(函数 analyze_string)、稀疏矩阵阵列(函数 analyze_sparse)、单精度数值阵列(函数 analyze_single)和双精度数值阵列(函数 analyze_double)的方法。本例通过 mxGetClassID 函数得到输入 Matlab 阵列的类型,然后再调用相应的 analyze_XXXX 分析函数分析 Matlab 阵列的基本属性并显示其内容。读者可以参照本例的做法,完成 Matlab 整型数值阵列的分析。本实例的程序代码如下所示。

```
/ * showarray.c 文件内容 * /
#include <stdio.h>
#include <string.h>
#include "mex.h"

void          display_subscript(const mxArray * array_ptr,mwIndex index);
void          get_characteristics(const mxArray    * array_ptr);
mxClassID     analyze_class(const mxArray * array_ptr);

/ *  cell 数组分析函数 * /
static void analyze_cell(const mxArray * cell_array_ptr)
{
```

```
    mwSize          total_num_of_cells;
    mwIndex          index;
    const mxArray * cell_element_ptr;

    total_num_of_cells = mxGetNumberOfElements(cell_array_ptr);
    mexPrintf("Cell 数组的元素个数 = %d\n",total_num_of_cells);
    mexPrintf("\n");


    for (index = 0; index<total_num_of_cells; index++)
    {
        mexPrintf("\n\n\t\t 元组的元素为 ");
        display_subscript(cell_array_ptr,index);
        mexPrintf("\n");
        cell_element_ptr = mxGetCell(cell_array_ptr,index);
        if (cell_element_ptr == NULL)
        {
            mexPrintf("\t 空元组\n");
        }
        else
        {
          mexPrintf("------------------------------------------\n");
            get_characteristics(cell_element_ptr);
            analyze_class(cell_element_ptr);
            mexPrintf("\n");
        }
    }
      mexPrintf("\n");
}

static void analyze_structure(const mxArray * structure_array_ptr)
{
  mwSize total_num_of_elements;
  mwIndex index;
  int number_of_fields,field_index;
  const char    * field_name;
  const mxArray * field_array_ptr;

  mexPrintf("\n");
  total_num_of_elements = mxGetNumberOfElements(structure_array_ptr);
  number_of_fields = mxGetNumberOfFields(structure_array_ptr);

  /* 遍历结构体数组中的所有结构体 */
```

```
    for (index=0; index<total_num_of_elements; index++)  {

        /* 遍历当前结构体的所有域 */
        for (field_index=0; field_index<number_of_fields; field_index++)  {
            mexPrintf("\n\t\t");
            display_subscript(structure_array_ptr,index);
                field_name=mxGetFieldNameByNumber(structure_array_ptr,
                                                        field_index);
            mexPrintf(". %s\n",field_name);
            field_array_ptr=mxGetFieldByNumber(structure_array_ptr,
                                    index,
                                    field_index);
            if (field_array_ptr==NULL)
            {
                    mexPrintf("\tEmpty Field\n");
            }
            else
            {
                mexPrintf("------------------------------------------------\n");
                get_characteristics(field_array_ptr);
                analyze_class(field_array_ptr);
                mexPrintf("\n");
            }
        }
            mexPrintf("\n\n");
    }


}



/* 字符串数组分析函数 */
static void analyze_string(const mxArray * string_array_ptr)
{
        char * buf;
        mwSize   number_of_dimensions;
        const mwSize   * dims;
        mwSize   buflen,d,page;
        mwSize   total_number_of_pages,elements_per_page,nStringGetFlag;

        /* 分配待转换字符串需要的空间 */
        buflen=(mxGetNumberOfElements(string_array_ptr) + 1) * sizeof(mxChar);
```

```
buf = (char * )mxCalloc(buflen,sizeof(char));

printf("1");
if ((nStringGetFlag = mxGetString(string_array_ptr,buf,buflen)) ! = 0)
{
        if(nStringGetFlag == 1)
        {
            mexErrMsgTxt("分配的字符转换空间不足！\n");
            //mexWarnMsgTxt("分配的字符转换空间不足！\n");

        }
        else
        {
            mexErrMsgTxt("输入的数组不是字符数组,无法转换！\n");
        }
        return;
}

printf("2");
dims = mxGetDimensions(string_array_ptr);
number_of_dimensions = mxGetNumberOfDimensions(string_array_ptr);

/* 计算字符串数组每页的字符个数 */
elements_per_page = dims[0] * dims[1];

/* 计算字符串数组页数总和 */
total_number_of_pages = 1;
for (d = 2; d<number_of_dimensions; d++)
{
      total_number_of_pages * = dims[d];
}

 printf("3");

for (page = 0; page < total_number_of_pages; page++)
{
      mwSize row;

      for (row = 0; row<dims[0]; row++)
      {
            mwSize column;
            mwIndex index = (page * elements_per_page) + row;
            mexPrintf("\t");
```

```
                    display_subscript(string_array_ptr, index);
                    mexPrintf(" ");

                    for (column = 0; column<dims[1]; column++)
                    {
                            mexPrintf("%c",buf[index]);
                            index += dims[0];
                    }
                    mexPrintf("\n");
                }
            }
        printf("4");
}

/* 分析稀疏数组(Sparse Array)函数 */
static void analyze_sparse(const mxArray * array_ptr)
{
        double     *pr, *pi;
        mwIndex      *ir, *jc;
        mwSize      col, total = 0;
        mwIndex      starting_row_index, stopping_row_index, current_row_index;
        mwSize      n;

        /* Get the starting positions of all four data arrays. */
        pr = mxGetPr(array_ptr);
        pi = mxGetPi(array_ptr);
        ir = mxGetIr(array_ptr);
        jc = mxGetJc(array_ptr);

        /* Display the nonzero elements of the sparse array. */
        n = mxGetN(array_ptr);
        for (col = 0; col<n; col++)
        {
                starting_row_index = jc[col];
                stopping_row_index = jc[col + 1];
                if (starting_row_index == stopping_row_index)
                {
                        continue;
                }
                else
                {
                        for (   current_row_index = starting_row_index;
                                        current_row_index < stopping_row_index;
```

```
                                    current_row_index ++ )
                    {
                            if (mxIsComplex(array_ptr))
                            {
                                    mexPrintf("\t(%d,%d) = %g+ %g i\n",
                                                ir[current_row_index] + 1,
                                                    col + 1,pr[total],pi[total ++ ]);
                        }
                        else
                        {
                                mexPrintf("\t(%d,%d) = %g\n",ir[current_row_index] + 1,
                                                    col + 1,pr[total ++ ]);
                        }
                    }
                }
            }
}

/*单精度型数组的分析函数*/
static void analyze_single(const mxArray * array_ptr)
{
        float * pr, * pi;
        mwSize      total_num_of_elements;
        mwIndex index;

        pr = (float * )mxGetData(array_ptr);
        pi = (float * )mxGetImagData(array_ptr);
        total_num_of_elements = mxGetNumberOfElements(array_ptr);

        for (index = 0; index<total_num_of_elements; index ++ )
        {
                mexPrintf("\t");
                display_subscript(array_ptr,index);
                if (mxIsComplex(array_ptr))
                {
                        mexPrintf(" = %g  +  %gi\n", * pr ++ , * pi ++ );
                }
                else
                {
                        mexPrintf(" = %g\n", * pr ++ );
                }
        }
}
```

```
/* 双精度型数组的分析函数 */
static void analyze_double(const mxArray * array_ptr)
{
        double * pr, * pi;
        mwSize      total_num_of_elements;
        mwIndex index;

        pr = mxGetPr(array_ptr);
        pi = mxGetPi(array_ptr);
        total_num_of_elements = mxGetNumberOfElements(array_ptr);

        for (index = 0; index<total_num_of_elements; index++)
        {
                mexPrintf("\t");
                display_subscript(array_ptr, index);
                if (mxIsComplex(array_ptr))
                {
                        mexPrintf(" = %g + %gi\n", * pr++, * pi++);
                }
                else
                {
                        mexPrintf(" = %g\n", * pr++);
                }
        }
}

static void tryityouself()
{
        mexPrintf("请参考 single 数组的分析方法\n");
}

/* 所有数值(numeric)类型数组分析函数的统一入口 */
static void analyze_full(const mxArray * numeric_array_ptr)
{
        mxClassID    category;
        category = mxGetClassID(numeric_array_ptr);
        switch (category)
        {
            case mxINT8_CLASS:    tryityouself();    break;
            case mxUINT8_CLASS:   tryityouself();    break;
            case mxINT16_CLASS:   tryityouself();    break;
            case mxUINT16_CLASS: tryityouself(); break;
```

```
            case mxINT32_CLASS:   tryityouself();   break;
            case mxUINT32_CLASS: tryityouself(); break;
            case mxSINGLE_CLASS: analyze_single(numeric_array_ptr); break;
            case mxDOUBLE_CLASS: analyze_double(numeric_array_ptr); break;
    }
}

/* 显示响应元素的下标,输入的 index 是将数组看做一维线性的形式的形式下给出的 */
void display_subscript(const mxArray * array_ptr,mwIndex index)
{
        mwSize        inner,subindex,total,d,q,number_of_dimensions;
        mwSize             * subscript;
        const mwSize * dims;

        number_of_dimensions = mxGetNumberOfDimensions(array_ptr);
        subscript = mxCalloc(number_of_dimensions,sizeof(mwSize));
        dims = mxGetDimensions(array_ptr);

        mexPrintf("(");
        subindex = index;
        for(d = number_of_dimensions - 1; d< = 0 ; d-- )
            {
                for (total = 1,inner = 0; inner<d; inner++ )
                {
                    total * = dims[inner];
                }

                subscript[d] = subindex / total;
                subindex = subindex % total;
              /* if (d <= 0)
                {
                break;
                } */
        }

        for (q = 0; q<number_of_dimensions - 1; q++ )
        {
          mexPrintf(" %d,",subscript[q] + 1);
        }
        mexPrintf(" %d)",subscript[number_of_dimensions - 1] + 1);

        mxFree(subscript);
 }
```

```
/* 分析 mxArray 的各维数大小、类型等特征 */
void get_characteristics(const mxArray * array_ptr)
{
    const char      * name;
    const char      * class_name;
    const mwSize      * dims;
    char      * shape_string;
    char      * temp_string;
    mwSize      c;
    mwSize      number_of_dimensions;
    size_t      length_of_shape_string;

    /* 维数及各维数的大小 */
    number_of_dimensions = mxGetNumberOfDimensions(array_ptr);
    dims = mxGetDimensions(array_ptr);

    /* 多分配用于显示的字符串是为了各个维数之间用 X 相连 */
    shape_string = (char * )mxCalloc(number_of_dimensions * 3,sizeof(char));
    shape_string[0] = '\0';
    temp_string = (char * )mxCalloc(64,sizeof(char));

    for (c=0; c<number_of_dimensions; c++)
    {
      sprintf(temp_string," %dx",dims[c]);
      strcat(shape_string,temp_string);
    }

    length_of_shape_string = strlen(shape_string);
    /* 替换最后的一个 x,将其改为\0,字符串结束 */
    shape_string[length_of_shape_string - 1] = '\0';
    if (length_of_shape_string > 16)
    {
      sprintf(shape_string," %d - D\0",number_of_dimensions);
    }

    mexPrintf("维数: %s\n",shape_string);

    class_name = mxGetClassName(array_ptr);
    mexPrintf("类型: %s%s\n",class_name,mxIsSparse(array_ptr) ? " (sparse)" : "");

    mexPrintf("---------------------------------------------\n");
```

```
        mxFree(shape_string);
}

/* 根据 mxArray 的类型选择恰当的分析函数进行分析 */
mxClassID analyze_class(const mxArray * array_ptr)
{
    mxClassID   category;

    category = mxGetClassID(array_ptr);

    if (mxIsSparse(array_ptr))
    {
        analyze_sparse(array_ptr);
    }
    else
    {
        switch (category)
        {
            case mxCHAR_CLASS:      analyze_string(array_ptr);      break;
            case mxSTRUCT_CLASS:    analyze_structure(array_ptr);   break;
            case mxCELL_CLASS:      analyze_cell(array_ptr);        break;
            case mxUNKNOWN_CLASS: mexWarnMsgTxt("Unknown class."); break;
            default:                analyze_full(array_ptr);        break;
        }
    }

    return(category);
}

/* mexFunction */
void mexFunction( int nlhs,mxArray * plhs[],int nrhs,const mxArray * prhs[])
{
        int  i;
        int nFlag;
        if(nrhs>1)
        {
            nFlag = (int)mxGetScalar(prhs[nrhs - 1]);/* 是否显示详细信息 */
        }
        else
        {
            nFlag = 1;
        }
        for (i=0; i<(((nrhs - 1)>1)? (nrhs - 1):1); i++)
```

```
        {
                mexPrintf("\n\n");
                mexPrintf("----------------------第%d个输入参数------------------\n",i);
                mexPrintf("名称：%s%d%c\n","prhs[",i,']');
                get_characteristics(prhs[i]);
                if(nFlag)
                {
                        analyze_class(prhs[i]);
                }
        }
}
```

Matlab 测试代码为：

```
sp=zeros(10,10);
sp(4,4)=1;sp(5,5)=1;sp(1,6)=1;
st.f1=sp;st.f2='hello';st.f3=1:100;
d=rand(5,5);
showarray(sp,st,d,1);
showarray(sp,st,d,0);
```

3.14.3　向 MAT 文件中写入 mxArray 变量

MAT 文件是 Matlab 特有的文件类型，将数据保存位 MAT 文件格式对于 Matlab 与其他开发环境交换数据十分方便。Matlab mat API 函数提供对 MAT 文件的操作接口。下面的实例说明如何将将 mxArray 变量写入 MAT 文件中。

```
/* matWritemxArray.c 文件内容 */
#include "memory.h"
#include "mat.h"

#ifndef NULL
#define NULL 0
#endif


int main(int argc,char * argv[])
{
    MATFile * pmat;
    mxArray * pa1, * pa2, * pa3;
    double * pdata=NULL;
    double data[9]={ 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0 };
    const char * file="mattest.mat";
    char * * dir;
    int ndir;
```

```
int status,i;

printf("正在创建文件 %s...\n\n",file);
pmat = matOpen(file,"w");
if (pmat == NULL)
{
    printf("文件 %s 创建出错\n",file);
    printf("请检查是否具有当前目录的写文件权限或者当前磁盘是否还有剩余空间？\n");
    return -1;
}

pa1 = mxCreateDoubleMatrix(3,3,mxREAL);
if (pa1 == NULL)
{
    printf("%s : 内存分配出错 %d 行\n",__FILE__,__LINE__);
    printf("无法创建需要的 mxArray 数组.\n");
    return -1;
}

pdata = mxGetPr(pa1);
for(i=0;i<mxGetM(pa1) * mxGetN(pa1);i++)
{
    pdata[i] = mxGetM(pa1) * mxGetN(pa1) - i + 1;
}

pa2 = mxCreateDoubleMatrix(3,3,mxREAL);
if (pa2 == NULL)
{
    printf("%s : 内存分配出错 %d 行\n",__FILE__,__LINE__);
    printf("无法创建需要的 mxArray 数组.\n");
    return -1;
}
memcpy((void *)(mxGetPr(pa2)),(void *)data,sizeof(data));

pa3 = mxCreateString("Matlab: the language of technical computing");
if (pa3 == NULL)
{
    printf("%s : 内存分配出错 %d 行\n",__FILE__,__LINE__);
    printf("无法创建需要的 mxArray 数组.\n");
    return -1;
}

status = matPutVariable(pmat,"LocalDouble",pa1);
```

```
if (status ! = 0)
{
    printf(" %s :  matPutVariable 函数出错,%d 行\n",    __FILE__,__LINE__);
    return -1;
}

status=matPutVariableAsGlobal(pmat,"GlobalDouble",pa2);
if (status ! = 0)
{
    printf(" %s: matPutVariableAsGlobal 出错,%d 行\n",__FILE__,__LINE__);
    return -1;
}

status=matPutVariable(pmat,"LocalString",pa3);
if (status ! = 0)
{
    printf(" %s :  matPutVariable 函数出错,%d 行\n",    __FILE__,__LINE__);
    return -1;
}


mxDestroyArray(pa1);
mxDestroyArray(pa2);
mxDestroyArray(pa3);

printf("文件 %s 创建成功! \n\n",file);

if (matClose(pmat) ! = 0)
{
    printf("关闭 %s 文件出错! \n",file);
    return -1;
}

/*重新打开生成的 MAT 文件,并将其中的变量显示出来*/
pmat=matOpen(file,"r");
if (pmat==NULL)
{
    printf("打开文件 %s 出错! \n",file);
    return -1;
}

dir=matGetDir(pmat,&ndir);
if (dir==NULL)
```

```
    {
        printf("MAT 文件 %s 没有变量存在! \n",file);
        return(1);
    }
    else
    {
        printf("MAT 文件%s 中的变量有:\n",file);
        for (i=0; i < ndir; i++ )
        {
            printf("\t%s\n",dir[i]);
        }
    }

    if (matClose(pmat) ! = 0)
    {
        printf("关闭 %s 文件出错! \n",file);
        return(EXIT_FAILURE);
    }
    getchar();
    return 0;
}
```

采用命令:

mbuild -f <matlabroot>\bin\win32\mexopts\msvc60engmatopts. bat matWritemxArray. c

将 matWritemxArray. c 编译为可执行文件(其中〈matlabroot〉表示 Matlab 安装的根目录)。此程序的运行结果如图 3-9 所示。

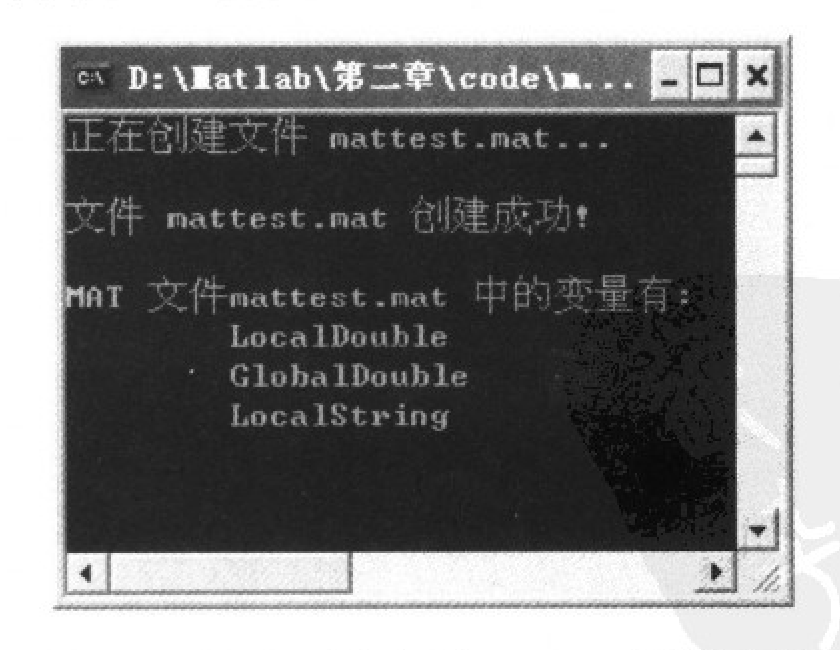


图 3-9　向 MAT 文件中写入 mxArray 变量的实例

3.14.4　从 MAT 文件中读取 mxArray 变量

如果想将 Matlab 开发环境的数据输出到其他开发环境中,可以通过直接将 Matlab 阵列

数据存储为文本文件及二进制文件等通用的文件格式。但是,如果需要输出的数据存放在MAT 文件中,就需要采用相应的以“mat-”开头的 API 函数将数据从 MAT 文件中以 mxArray 变量的方式读出来,然后再转换位开发者需要的数据格式。下面的实例代码演示的是从MAT 文件中读入 Matlab 阵列变量的方法,读入的 Matlab 变量采用 mxArray 类型来表示。

```
/* matreadmxArray.c 文件内容 */
#include <stdio.h>
#include "string.h"
#include "mat.h"

#define _FILE_NAME_LEN 100

int analyze_matfile(const char * file)
{
    MATFile * pmat;
    char ** dir;
    const char * name;
    int ndir;
    int i;
    mxArray * pa;

    mexPrintf("开始读取 MAT 文件 %s...\n\n",file);

    pmat = matOpen(file,"r");
    if (pmat == NULL)
    {
        mexPrintf("文件%s打开出错! \n",file);
        return(1);
    }


    dir = matGetDir(pmat,&ndir);
    if (dir == NULL)
    {
        mexPrintf("当前 MAT 文件%s中没有任何变量! \n",file);
        return(1);
    }
    else
    {
        mexPrintf("MAT 文件%s的变量为:\n",file);
        for (i=0; i< ndir; i++)
        {
            mexPrintf("%s\n",dir[i]);
```

```
        }
    }
    mxFree(dir);

    if (matClose(pmat) ! = 0)
    {
        mexPrintf("关闭文件 %s 出错! \n",file);
        return(1);
    }

    /* 重新打开 Matlab MAT 文件 */
    pmat = matOpen(file,"r");
    if (pmat == NULL)
    {
        mexPrintf("打开文件 %s 出错! \n",file);
        return(1);
    }

    for (i=0; i < ndir; i++)
    {
        pa = matGetNextVariableInfo(pmat,&name);
        if (pa == NULL)
        {
            mexPrintf("读取文件 %s 出错! \n",file);
            return(1);
        }

        mexPrintf("Matlab 数组 %s 的:\n\t 维数为: %d\n",
                              name,mxGetNumberOfDimensions(pa));
        mexPrintf("                    \t 行数 x 列数为 %dx%d\n",
                              mxGetM(pa),mxGetN(pa));

        mxDestroyArray(pa);
    }


    if (matClose(pmat) ! = 0)
    {
        mexPrintf("关闭文件 %s 出错! \n",file);
        return(1);
    }
    mexPrintf("文件%s 分析完毕! \n",file);
    return(0);
```

```
}


int main(int argc,char * * argv)
{
    char name[_FILE_NAME_LEN];
    int num=0;
    int nFlag1,nFlag2;

    mexPrintf("请输入要读取的文件名称(*.mat):");
    scanf("%s",name);
    while((name[num++]!='\0')&&(num<=_FILE_NAME_LEN-1));
    num=num-1;
    if(num>=_FILE_NAME_LEN-5)
    {
        mexPrintf("输入的文件名太长!\n");
        return 0;
    }

    nFlag1=strcmp(name+num-4,".MAT");
    nFlag2=strcmp(name+num-4,".mat");

    if(nFlag1&&nFlag2)
    {
        name[num]='.';
        name[num+1]='M';
        name[num+2]='A';
        name[num+3]='T';
        name[num+4]='\0';
    }
    analyze_matfile(name);
    return 0;
}
```

在 Matlab 命令行中通过命令:

```
mbuild -f <matlabroot>\bin\win32\mexopts\msvc60engmatopts.bat matreadmxArray.c
```

编译上述程序(其中〈matlabroot〉表示 Matlab 安装的根目录),并运行 matreadmxArray.exe,执行结果如图 3-10 所示。

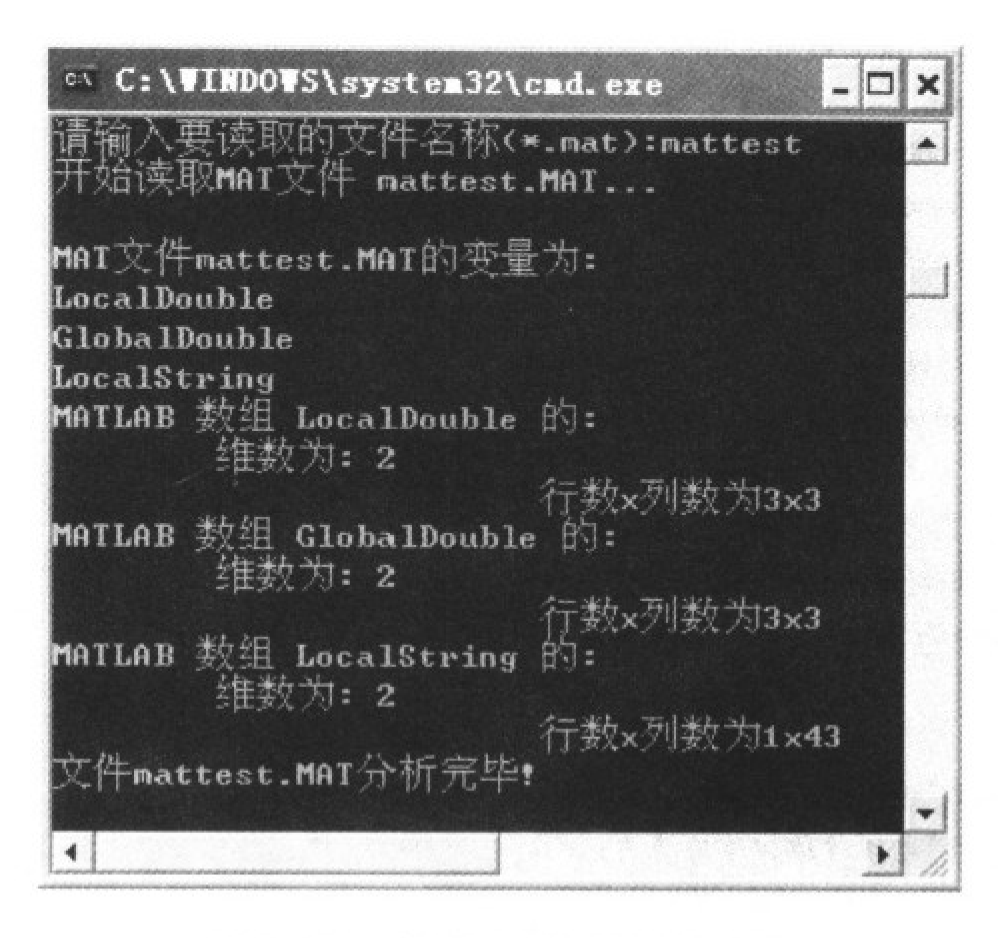


图 3-10 读 MAT 文件实例运行结果

3.14.5 通讯录(结构体和 MAT 文件)

下面的实例采用 MAT 文件作为存储的文件格式,采用 Matlab 结构体作为通讯录的主要数据结构,实现了一个简单的通讯录。通讯录采用文件"phonebook.mat"作为存储文件,采用拥有如下域的结构体作为主要数据结构。

- name 姓名;
- addr 住址;
- phonenum 电话。

通讯录的程序如下所示。

```
/* phonebook.c 文件内容 */
#include "mex.h"
#include "mat.h"
#include "string.h"

#define _MAX_NUM_OF_ELEMENTS 3000

/* 注意文件指针的操作 */
void mexFunction( int nlhs,mxArray * plhs[],
                  int nrhs,const mxArray * prhs[] )
{
    /* 检查参数的有效性 */
    int i,j;
    int nfields,nRecordNum,nUsedNum;
```



```
mwSize nStructElementNum;
MATFile * pmat;
mxArray *  pPhoneBook = NULL;
mxArray *  pElement;
mxArray *  pTmpField;
char tempstring[300];
char name[] = "phonebook";
char file[] = "phonebook.mat";
int status = 0;
int fieldnum[3];
char fieldStr[200];/* 存放读取的变量名称 */
int isNewFile = 0;
if(nrhs == 0)
{
    mexPrintf("请按照电话簿的格式输入一个结构体:\n");
    mexPrintf("电话簿结构体的域为:\n");
    mexPrintf("\t name 姓名\n");
    mexPrintf("\t addr 住址\n");
    mexPrintf("\t phonenum 电话\n");
    return;
}
else
{
    for(i = 0;i<nrhs;i++)
    {
        if(! mxIsStruct(prhs[i]))
        {
            mexPrintf("参数%d不是结构体! \n",i+1);
            return;
        }
    }
}

/* 读入电话簿 */
pmat = matOpen(file,"r");
if(pmat == NULL)
{
  mexPrintf("读取文件%s错误! \n",file);
    mexPrintf("创建文件%s...\n",file);
    /* 读取的文件不存在,重新创建"phonebook.mat"文件 */
    pmat = matOpen(file,"w");
    if(pmat == NULL)
    {
```

```
            mexPrintf("创建文件%s失败！\n",file);
            return;
        }
    else
        {
            matClose(pmat);
              pmat = matOpen(file,"u");
            if(pmat == NULL)
                {
                    mexPrintf("重新读取文件%s错误！\n",file);
                    return;
                }
            isNewFile = 1;
        }
}

pPhoneBook = matGetVariable(pmat,"phonebook");
if(pPhoneBook! = NULL)
{
    if(! mxIsStruct(pPhoneBook))
    {
        mexPrintf("文件%s中不包含电话簿信息！\n",file);
        matClose(pmat);
        return;
    }
}
else
{
    /*在新创建的phonebook.mat文件加入phonebook结构体阵列*/
    mxArray * pa;
    mwSize ndim = 2;
    mwSize dims[2] = {1,_MAX_NUM_OF_ELEMENTS};
    char field1[] = "name";
    char field2[] = "addr";
    char field3[] = "phonenum";
    char * field_names[3] = {field1,field2,field3};
    int i = 0;
    mexPrintf("文件%s中不包含电话簿信息！\n",file);
    mexPrintf("创建电话簿信息！\n");
    pa = mxCreateStructArray(ndim,dims,3,field_names);
    /*将pa的name域初始化为"-1"字符串*/
    for(i = 0;i<mxGetNumberOfElements(pa);i++)
        {
```

```
                mxSetFieldByNumber(pa,i,0,mxCreateString(" - 1"));
            }
        matPutVariable(pmat,"phonebook",pa);
        mxDestroyArray(pa);
        pa = NULL;
        pPhoneBook = matGetVariable(pmat,"phonebook");
    }

    nRecordNum = mxGetNumberOfElements(pPhoneBook);

    /*
            寻找当前未被占用的通讯录空间:
                    通讯录的最大条数可以根据
                    _MAX_NUM_OF_ELEMENTS 设定
    */
    i=0;
    while(i<nRecordNum)
    {
      pElement = mxGetField(pPhoneBook,i ,"name");
      mxGetString(pElement,tempstring ,300);
      /*
        如果结构体元素的 name 域的值为"-1"的话,
        当前结构体元素表示空的通讯录元素
       */
      if((*tempstring == '-')&&(*(tempstring + 1) == '1'))
      {
            break;
      }
      i++;
    }

    if(i == nRecordNum)
    {
        mexErrMsgTxt("电话簿已满! \n");
        matClose(pmat);
        return;
    }

    /*
            电话簿的域分别为:
                        name 姓名
                        addr   住址
                        phonenum 电话
```

```
*/
for(j=0;j<nrhs;j++)
{
        nStructElementNum=mxGetNumberOfElements(prhs[j]);
        if((nStructElementNum>1)||(nStructElementNum==0))
        {
                mexPrintf("参数%d与电话簿的格式不符\n",j+1);
                continue;
        }
        nfields=mxGetNumberOfFields(prhs[j]);
        if(nfields! =3)
        {
                mexPrintf("参数%d与电话簿的格式不符\n",j+1);
                continue;
        }

        pTmpField=mxGetField(prhs[j],0,"name");
        if(pTmpField==NULL)
        {
                printf("输入的第%d各变量的 name 域为空！\n",j);
                continue;
        }
        mxSetField(pPhoneBook,i+j, "name",mxDuplicateArray(pTmpField));
        pTmpField=mxGetField(prhs[j],0,"addr");
        if(pTmpField==NULL)
        {
                printf("输入的第%d各变量的 name 域为空！\n",j);
                continue;
        }
        mxSetField(pPhoneBook,i+j, "addr",mxDuplicateArray(pTmpField));
        pTmpField=mxGetField(prhs[j],0,"phonenum");
        if(pTmpField==NULL)
        {
                printf("输入的第%d各变量的 name 域为空！\n",j);
                continue;
        }
        mxSetField(pPhoneBook,i+j, "phonenum",mxDuplicateArray(pTmpField));
}

/*保存电话簿到*.mat文件中*/
matClose(pmat);
pmat=matOpen(file,"w");
status=matPutVariable(pmat,name,pPhoneBook);
```

```
    if(status! =0)
    {
            printf("变量%s写入文件%s出错! \n",name,file);
    }

    matClose(pmat);

    /*显示当前电话簿中的内容*/
    fieldnum[0] = mxGetFieldNumber(pPhoneBook,"name");
    fieldnum[1] = mxGetFieldNumber(pPhoneBook,"addr");
    fieldnum[2] = mxGetFieldNumber(pPhoneBook,"phonenum");

    nStructElementNum = mxGetNumberOfElements(pPhoneBook);
    for(i = 0;i<nStructElementNum;i++)
    {
        pTmpField = mxGetFieldByNumber(pPhoneBook,i,fieldnum[0]);
        mxGetString(pTmpField,fieldStr,200);
        if((*fieldStr == '-')&&(*(fieldStr + 1) == '1'))
        {
            break;
        }
        printf("第%d个联系人的资料:\n",i+1);
        printf("\t姓名:%s\n",fieldStr);

        pTmpField = mxGetFieldByNumber(pPhoneBook,i,fieldnum[1]);
        mxGetString(pTmpField,fieldStr,200);
        printf("\t地址:%s\n",fieldStr);

        pTmpField = mxGetFieldByNumber(pPhoneBook,i,fieldnum[2]);
        mxGetString(pTmpField,fieldStr,200);
        printf("\t联系电话:%s\n",fieldStr);
    }

    if(pPhoneBook! =NULL)
    {
        mxDestroyArray(pPhoneBook);
    }

    return;
}
```

Matlab 测试代码及测试结果如下所示。

```
>> mex phonebook.c
```

```
>> ly = struct('name','刘勇','addr','河北石家庄','phonenum','031161615053');
>> phonebook(ly)
读取文件 phonebook.mat 错误!
创建文件 phonebook.mat...
文件 phonebook.mat 中不包含电话簿信息!
创建电话簿信息!
第 1 个联系人的资料:
    姓名:刘勇
    地址:河北石家庄
    联系电话:031161615053
>> df = struct('name','东方','addr','河北石家庄','phonenum','031161615054');
>> phonebook(df)
第 1 个联系人的资料:
    姓名: 刘勇
    地址: 河北石家庄
    联系电话: 031161615053
第 2 个联系人的资料:
    姓名: 东方
    地址: 河北石家庄
    联系电话: 031161615054
>>
```

3.15　在 Visual C++中调试 MEX 文件

采用 MEX 文件的方式将 C 语言编写的代码嵌入到 Matlab 中执行是 Matlab 与 C 语言混合编程的一种重要方式。但是,MEX 文件不能在 Matlab 中像编写.m 文件那样便于调试,如果 MEX 文件实现的功能稍微复杂一点就会使程序的调试工作十分繁琐。尤其是对于很多已经习惯于采用 C/C++开发环境的开发人员来说,更是非常的不适应。由于 MEX 文件其实就是动态链接库,因而可以采用 Visual C++ 6.0 作为开发和调试 MEX 文件的工具。

通过 Visual C++ 6.0 建立可调试的 MEX 文件的步骤如下所述。

(1) 建立新的动态链接库工程

打开 Visual C++ 6.0,选择 File|New 菜单项,将会弹出如图 3-11 所示的对话框,选择 Win32 Dynamic-Link Library 选项,建立一个新的动态链接库工程。

(2) 建立新的 Matlab MEX 文件

单击 OK 按钮后,弹出如图 3-12 所示的对话框,选择"空的动态链接库"选项,创建一个新的 Matlab MEX 文件(在 Windows 下实际上是动态链接库文件),然后单击 Finish 按钮继续。

(3) 添加 testmexvc.c 文件

选择 Project|Add To Project|Files 菜单项,加入 testmexvc.c 文件。首先编辑 testmexvc.c 文件,其中 mexFucntion 必不可少。下面给出一个空的 mexFunction 的模板。

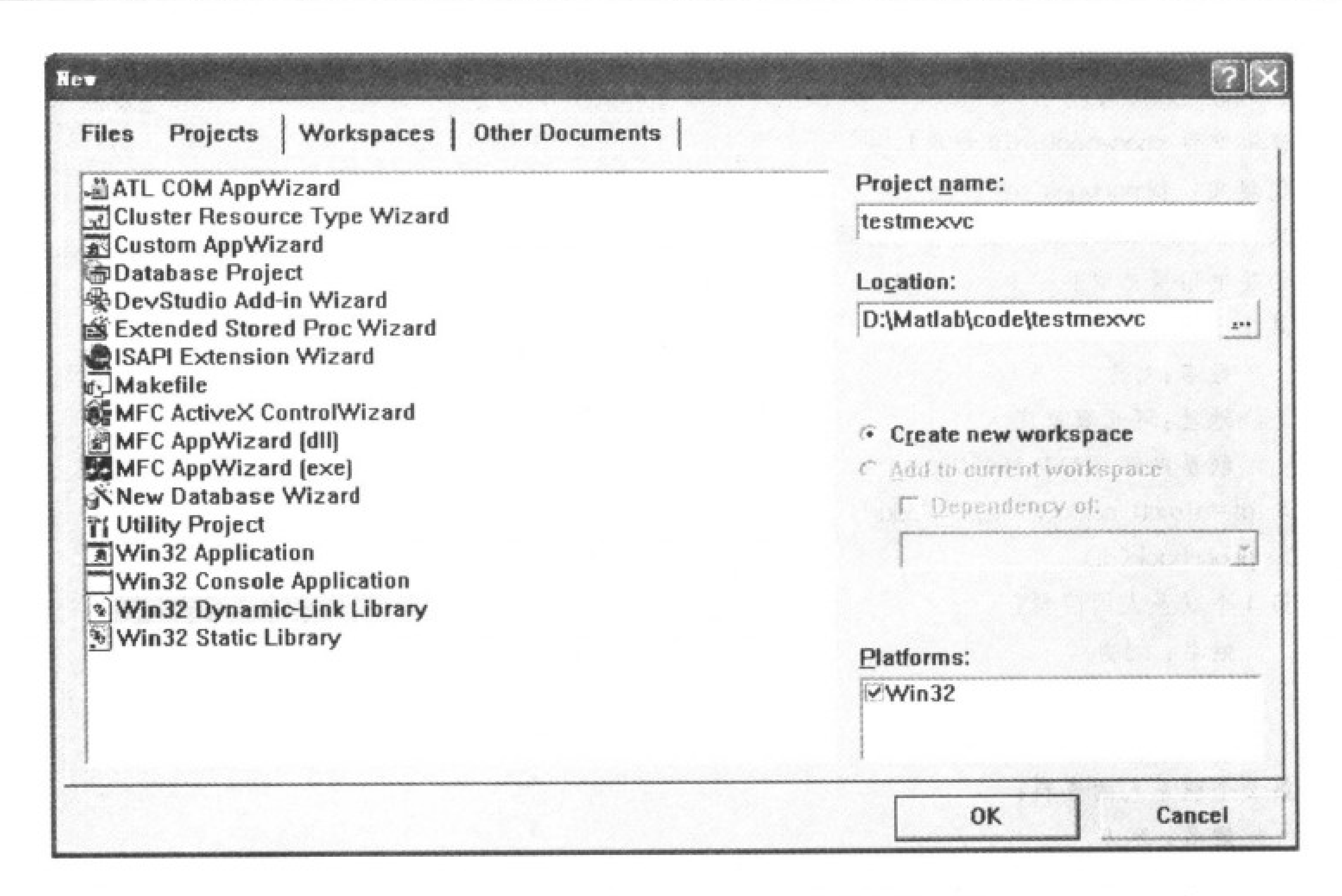


图 3-11　建立一个 Matlab MEX 文件的新工程

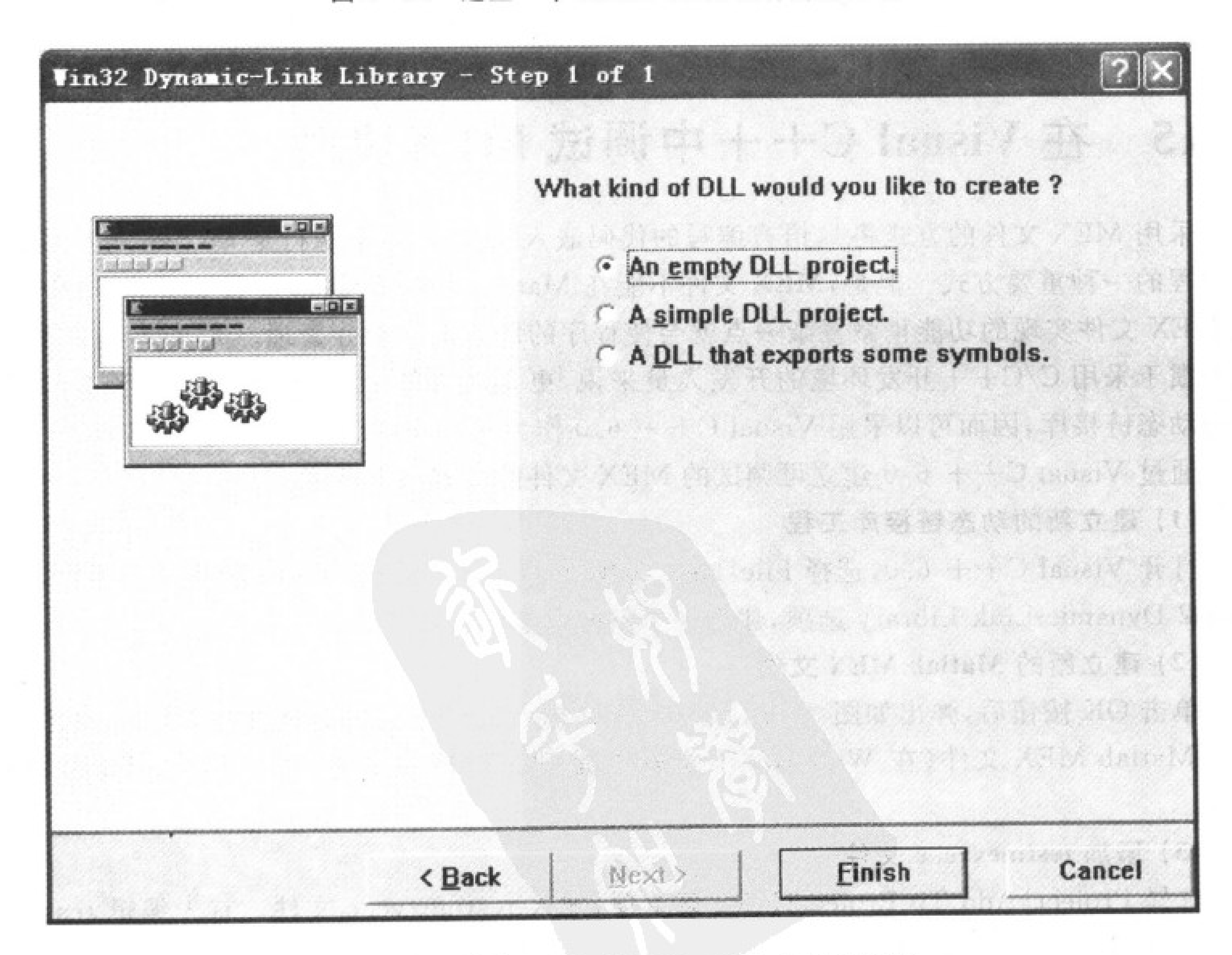


图 3-12　建立 MEX 文件 Visual C++工程选项

```
/* testmexvc.c 文件内容 */
#include "stdio.h" /* 可以省略 */
#include "mex.h"
void mexFunction(int nlhs,mxArray * plhs[],int nrhs,   const mxArray * prhs[])
{
    printf("Hello World\n");
    mexPrintf("Hello World\n");
}
```

(4) 建立 testmexvc.def 文件

用文本编辑器建立 testmexvc.def 文件。下面是一个标准的 *.def 文件模板,其大致含义是告诉编译器建立一个 testmexvc.mexw32 文件,并且其导出函数是 mexFunction。

```
LIBRARY testmexvc
EXPORTS mexFunction
```

testmexvc.def 文件建立完成以后,通过选择 Project|Add To Project 菜单项将其加入到已经建立的 Visual C++工程中,如图 3-13 所示。

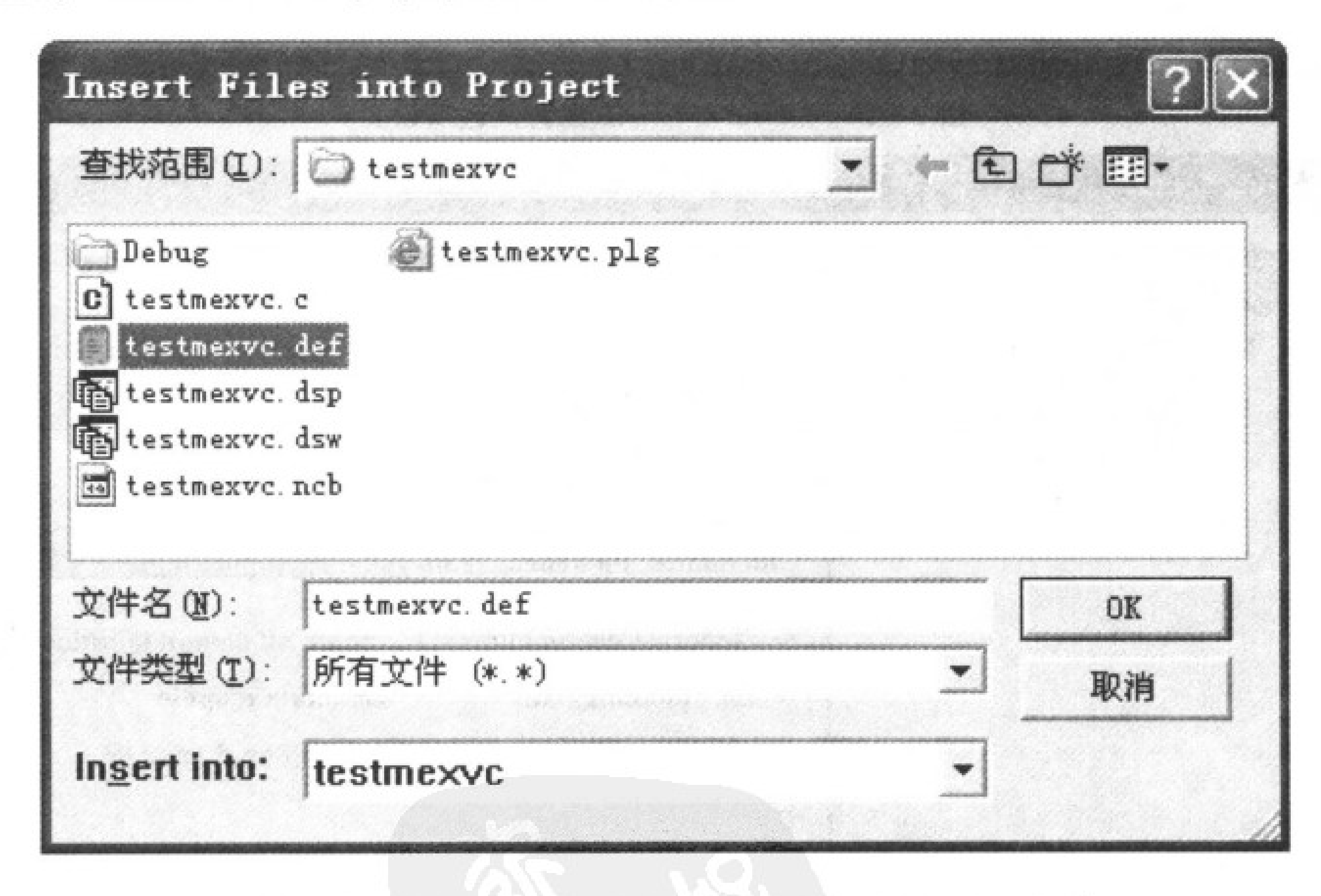


图 3-13 将 testmexvc.def 文件加入到 Visual C++工程中

(5) Visual C++工程设置

设置 Visual C++工程,以实现 MEX 文件的调试。

① 选择 Project|Setting 菜单项,弹出 Project Settings 对话框,如图 3-14 所示,选择 Debug 选项卡,在 Executable for debug session 文本框中选择所需的 Matlab 启动程序的地址。

② 在 Project Settings 对话框中选择 Link 选择卡,如图 3-15 所示,在 Object/library modules 文本框中输入需要加入的 Matlab 静态链接库,例如 libmx.lib、libmex.lib 和 libmat.lib。

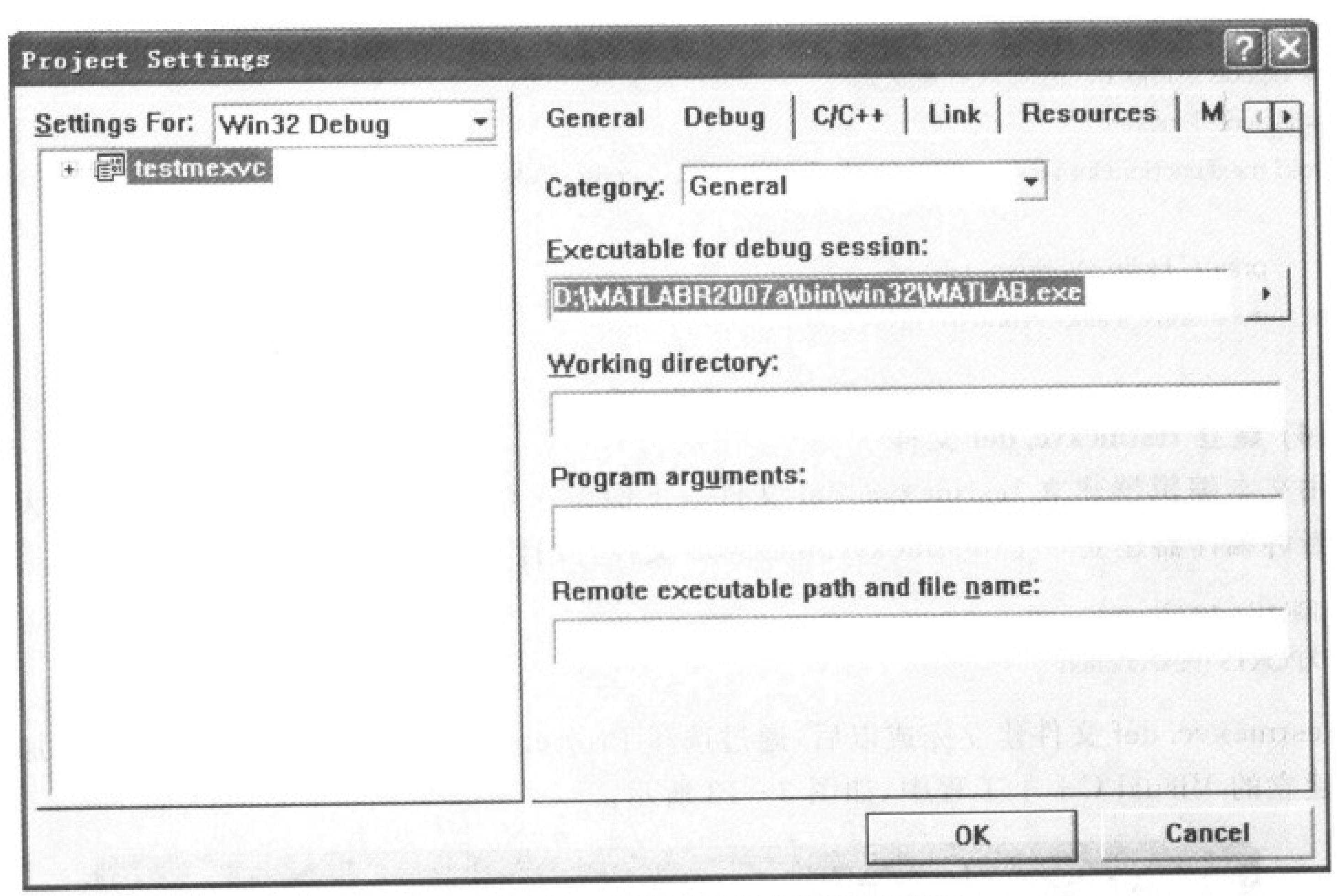


图 3-14 Visual C++工程调试选项设置

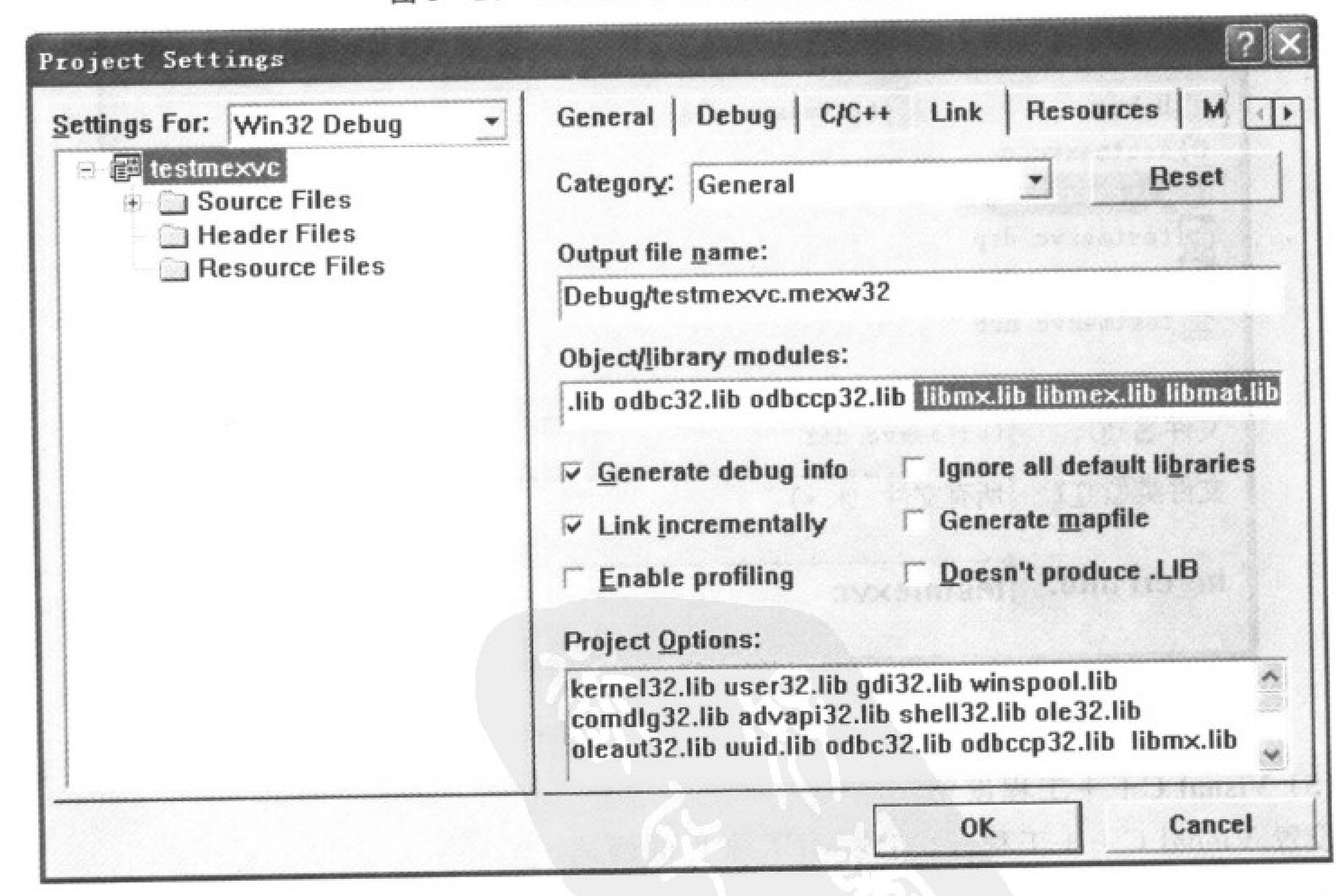


图 3-15 Visual C++工程 Link 选项卡设置

在 Output file name 文本框中更改输出文件的扩展名，如果是 32 位系统则改为 mexw32，如果是 64 位系统则改为 mexw64，如图 3-15 所示。

③ 在 Project Settings 对话框中选择 C/C++选项卡，在 Preprocessor Definitions 文本框

中加入宏定义 MATLAB_MEX_FILE，如图 3－16 所示。

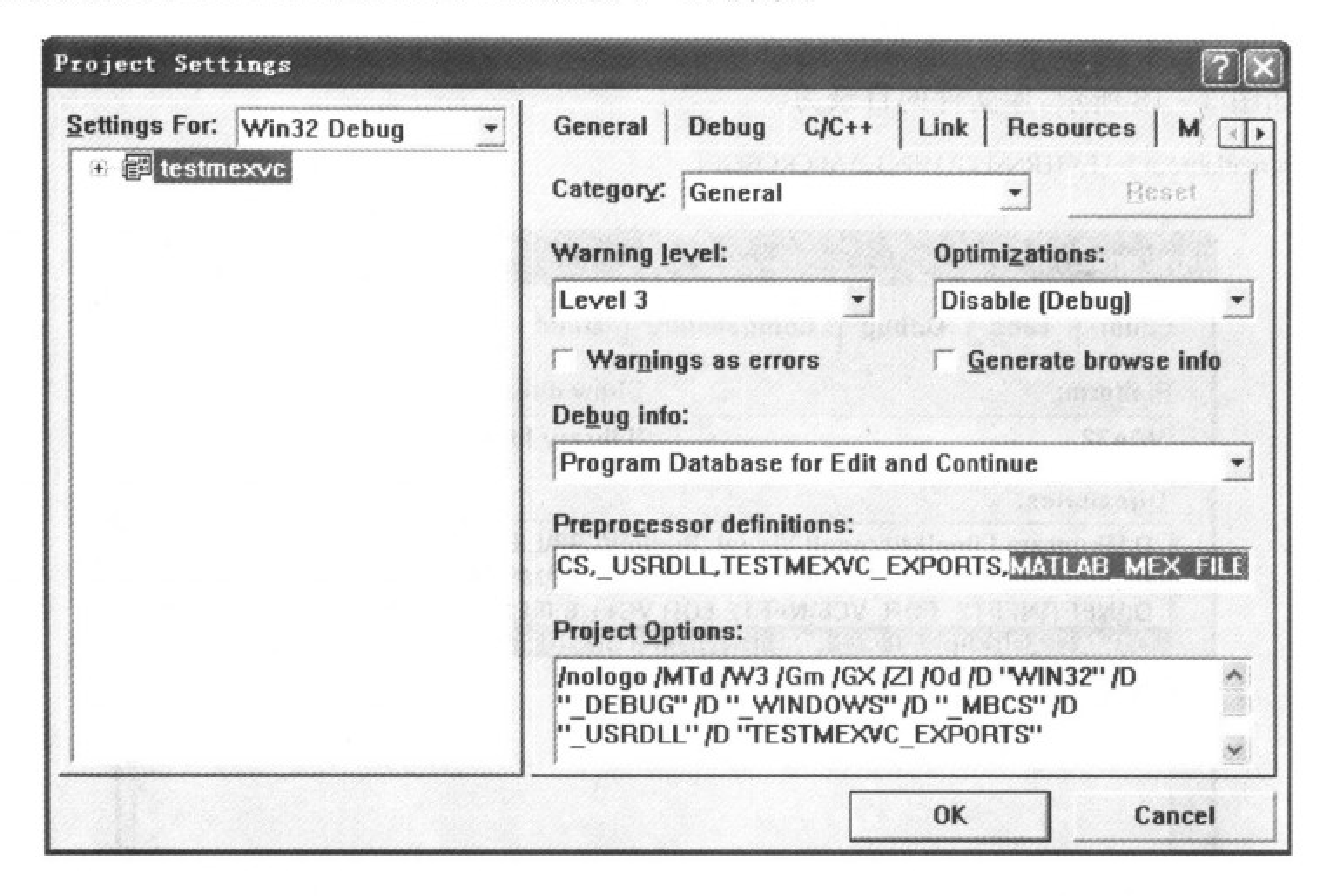


图 3－16　Visual C＋＋工程 C/C＋＋选项卡设置

(6) 设置 Visual C＋＋工程的头文件目录和静态链接库的目录

选择 Tools|Options 菜单项，设置 Visual C＋＋工程的头文件目录和静态链接库的目录，如图 3－17 和图 3－18 所示。其中头文件的目录为：

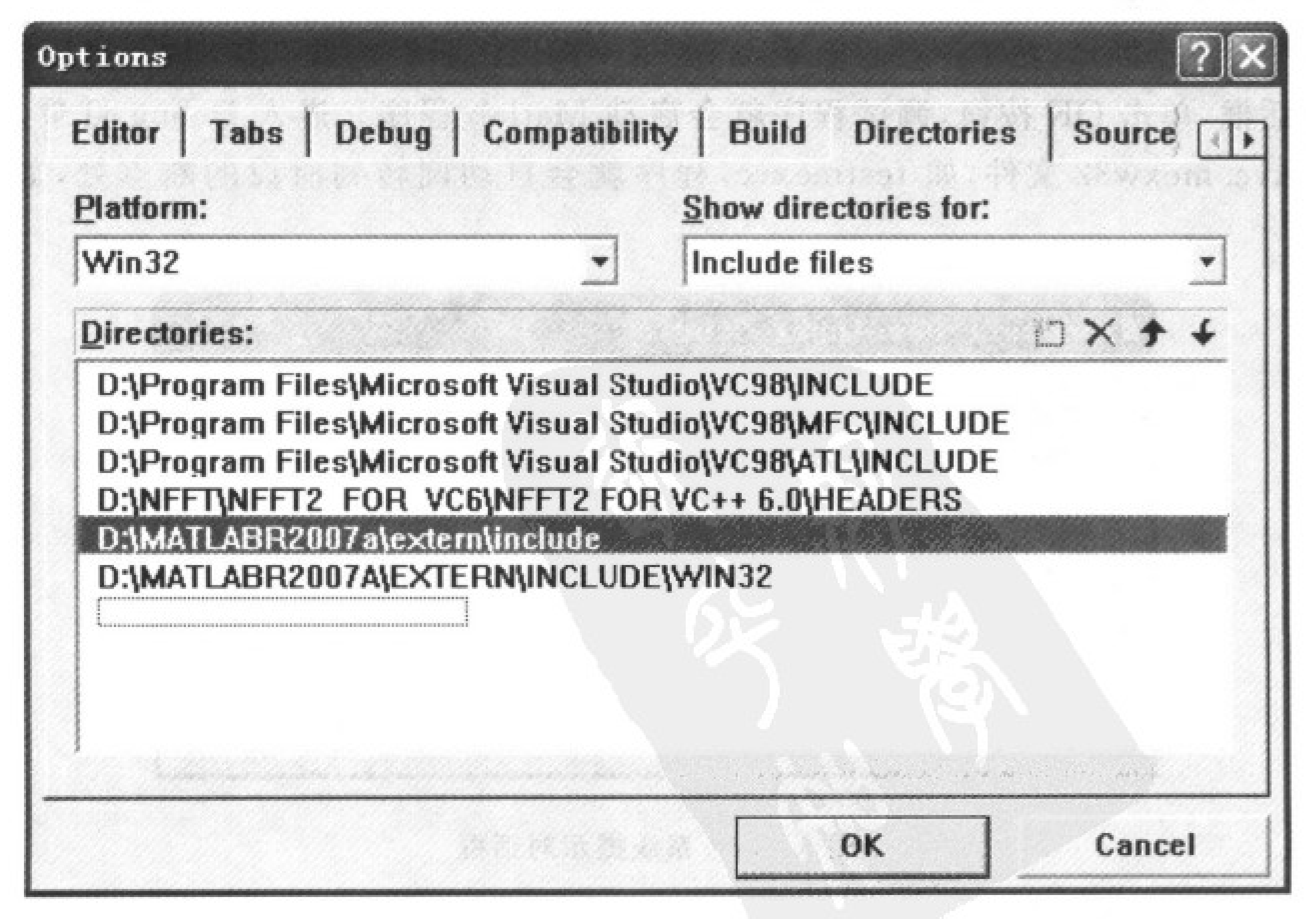


图 3－17　Visual C＋＋工程头文件目录的设置

<matlabroot>\extern\include

<matlabroot>\EXTERN\INCLUDE\WIN32

如图 3-18 所示，库文件的目录为：

<matlabroot>\EXTERN\LIB\WIN32\MICROSOFT

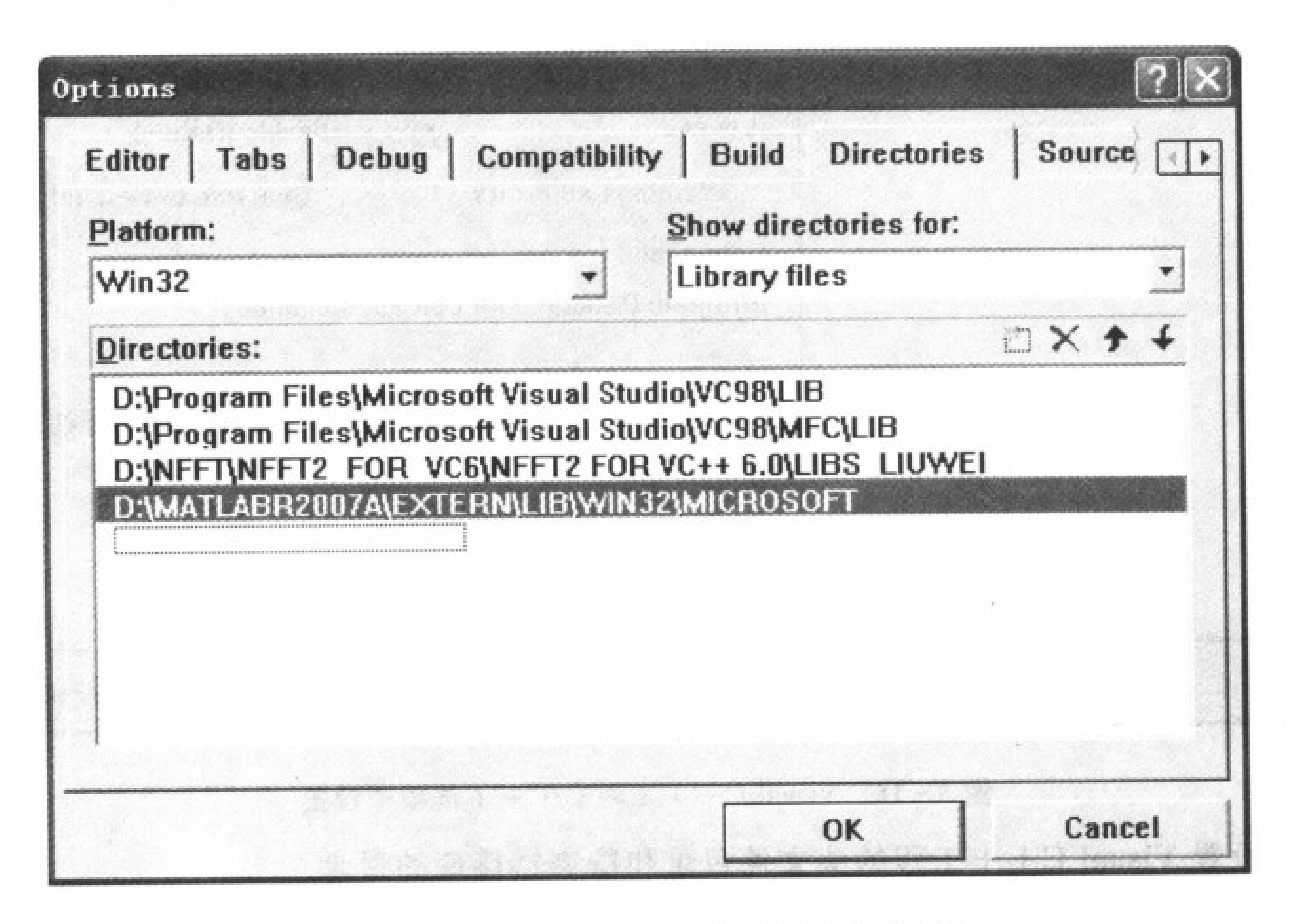


图 3-18 Visual C++工程库文件目录的设置

(7) 设置程序断点

为程序设一个断点，按 F5 键或者 Visual C++调试工具栏的调试按钮，会出现如图 3-19 所示的对话框，单击 OK 按钮，调试程序就会启动 Matlab 程序。进入 Debug 目录，执行生成的 testmexvc.mexw32 文件；如 testmexvc，程序就会自动调转到所设的断点处，如图 3-20 所示。

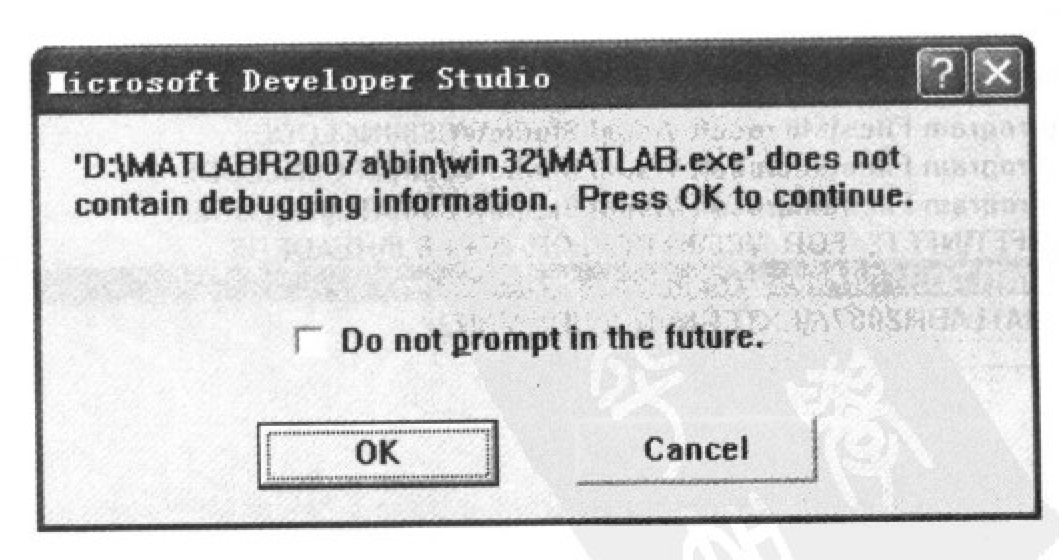


图 3-19 系统提示对话框

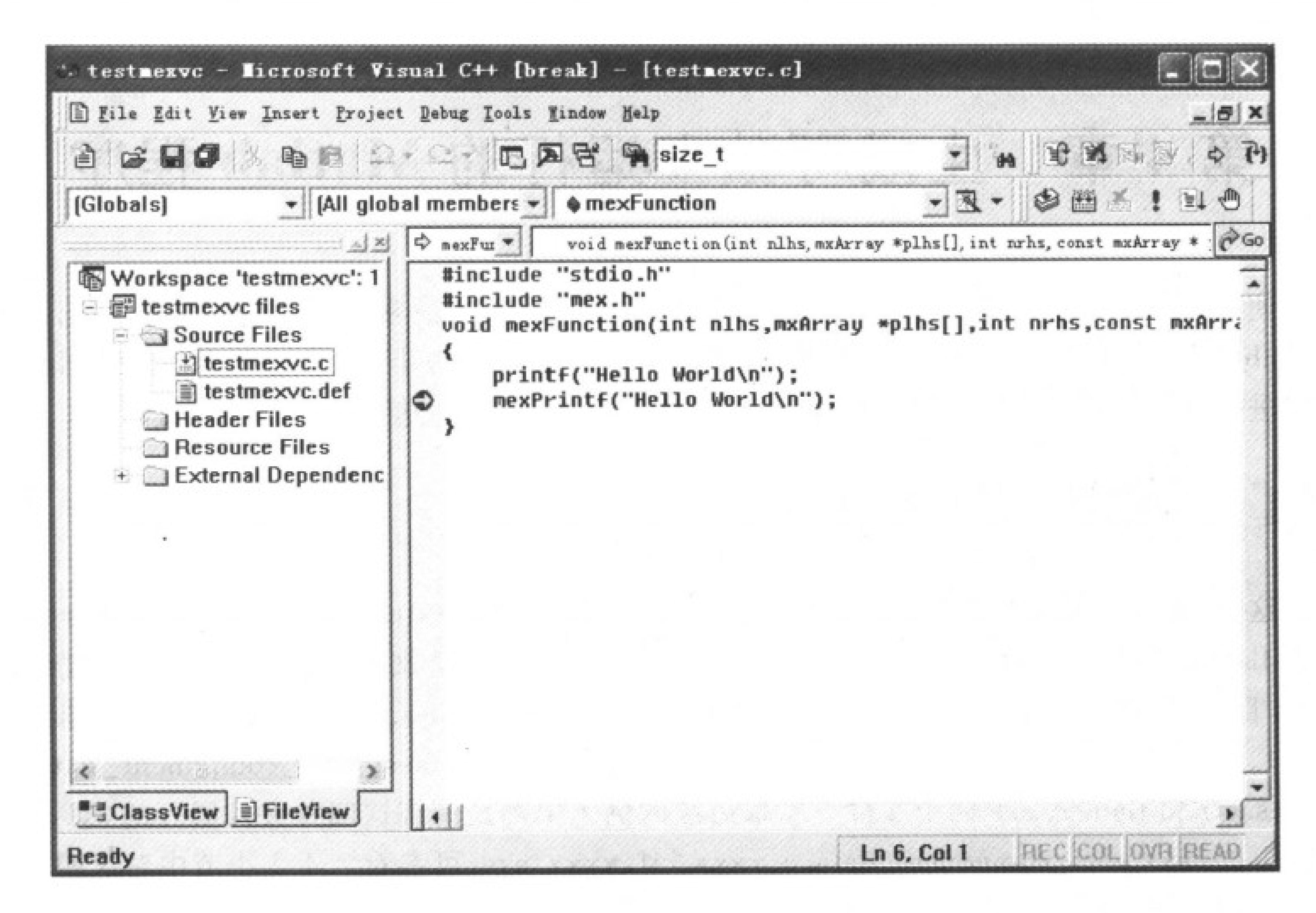


图 3-20　正在调试中的 Visual C++ MEX 工程

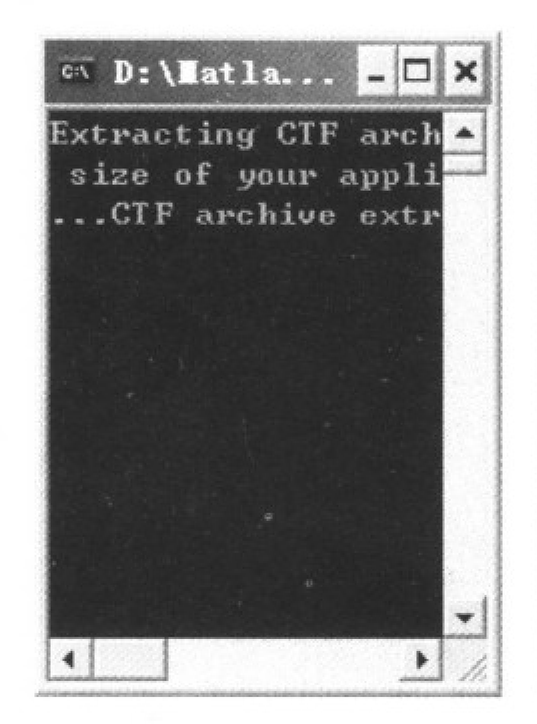


第 4 章　生成可独立运行的 Matlab 程序

使用 Matlab 编译器将 M 文件编译为独立可执行程序是最常用的一种方式。在第 2 章 Matlab 编译器中介绍了 Matlab 编译器编译独立可执行程序的步骤，通过本章的学习可以使读者进一步熟悉编译独立可执行 Matlab 程序的实用技巧，以期达到灵活运用的目的。

4.1　直接编译 M 文件

这是 Matlab 编译器最简单的应用，这里以〈matlabroot〉\extern\examples\compiler 目录下的 flames. m 文件为例（需要 flames. m 和 flames. mat 两个文件），将这两个文件复制到 work 目录下，然后用 mcc -m flames. m-a flames. mat 编译 flames. m 文件（运行结果如图 4－1 所示）。编译完成后，在目录下可以看到 flames_main. c、flames_mcc_component_data. c、flames. ctf 和 flames. exe 四个文件。本章后续的例子中都是围绕这四种文件展开的，即 xxxx _main. c、xxxx_mcc_component_data. c、xxxx. ctf、xxxx. exe（可参看 2. 4. 1 小节中对这四种文件进行的详细讲解）。



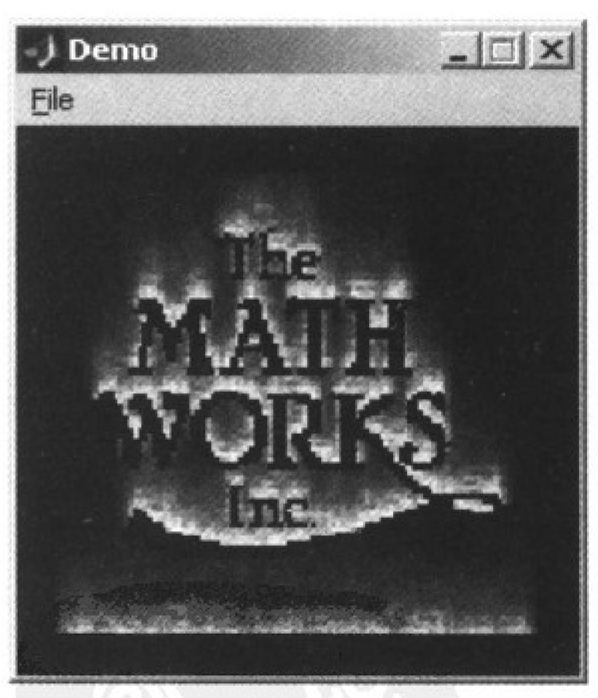


图 4－1　flames. exe 运行结果

4.2　Matlab M 文件中调用 C 函数

通过采用％＃external 语法结构，在 Matlab 函数中也可以调用 C 或 C＋＋函数。在下例中，假定在 Matlab 函数中需要调用一个与数据采集设备密切相关的底层 C 函数以获得该设备采集的数据。此时，可以首先用 M 文件完成整个程序，需要调用的 C 函数暂时用 M 函数代替，并用％＃external 标记。如下所示，整个 M 程序由 showdata 和 getdata 两个函数构成，

showdata 负责显示 getdata 从数据采集设备中得到的数据。

```
%以下代码在 showdata.m 文件中
function showdata
y=zeros(1,100); %将 y 分配为 1 x 100 的数组
for i=1:100
    y(i)=getdata;
end
plot(y);

%以下代码在 getdata.m 文件中
function y=getdata
%#external
%persistent 定义与 C 语言的 static 类似。采用 persistent 定义的变量在函数调
%用结束时，其仍然保存在内存中，下次函数调用时仍然可以读取其值
persistent t;
if (isempty(t))
     t=0;
end
     t=t + 0.05;
y=sin(t);
```

在命令行中键入 mcc -m showdata.m getdata.m 命令，将 showdata.m 及其关联的 getdata.m 编译为 C 文件，注意这里生成的 C 文件与 4.1.1 小节的文件不太相同。在生成的 C 文件中，多了一个 getdata_external.h 文件，其内容如下所示。

```
/*
 * Matlab Compiler: 4.6 (R2007a)
 * Date: Sun Jul 29 13:17:03 2007
 * Arguments: "-B" "macro_default" "-m" "-W" "main" "-T" "link:exe"
 * "showdata.m" "getdata.m"
 */

#ifndef __getdata_external_h
#define __getdata_external_h 1

#include "mclmcr.h"
#ifdef __cplusplus
extern "C" {
#endif

bool MW_CALL_CONV getdata(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[]);

#ifdef __cplusplus
```

```
}
#endif

#endif
```

这里的 getdata_external.h 实际上是一个接口文件。其中 getdata 函数声明为：

```
bool getdata(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[]);
```

getdata 便是要用 C 函数替换的 getdata 函数。此函数可以用 C 语言实现，命名在 getdata_external.c 文件中，笔者编写的 getdata_external.c 文件的内容如下所示。

```
/* 此程序不是自动生成，而是手工编写 */
#include "getdata_external.h"
bool MW_CALL_CONV getdata(int nlhs,mxArray * plhs[],int nrhs,mxArray * prhs[])
{
    /* 函数体的内容可以根据用户实际情况改变 */
    static double t=0.0;
    t=t + 0.05;
    plhs[0] = mxCreateDoubleScalar(sin(t));
}
```

完成 getdata_external.c 以后，通过命令：

```
mbuild showdata_main.c showdata_mcc_component_data.c getdata_external.c -output showdata.exe
```

将 showdata.m 文件编译为可执行文件，其运行结果如图 4-2 所示。

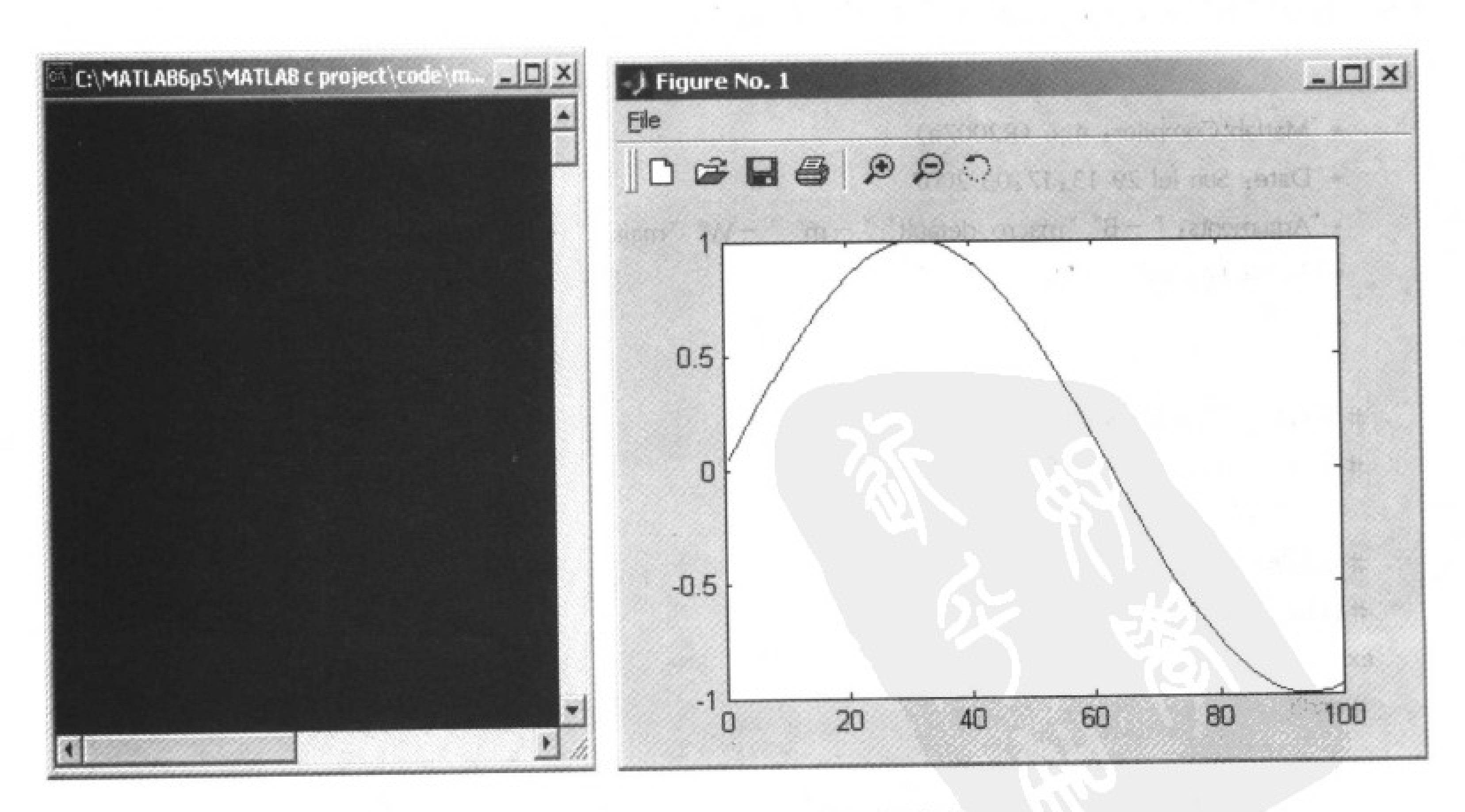


图 4-2　showdata.exe 的运行结果

4.3　在 C 语言中调用由 Matlab *.m 文件生成的函数

在 4.1.2 小节中，通过%#external 将 C 语言代码嵌入到 matlab *.m 文件中。在 C 语言函数中，也可以调用 *.m 函数，该类函数是经过 Matlab 编译器编译后的函数。下面以 mrandplot.m 函数为例，说明如何通过 C 语言调用 Matlab *.m 生成函数。

```
%以下程序在 mrandplot.m 文件中
function noise = mrandplot(n)
% function noise = mrandplot(n)
% mrandplot.m
noise = rand(n,1);
plot(noise);
```

1. 编译 mrand.m 文件

首先采用下面命令编译 mrandplot.m 文件：

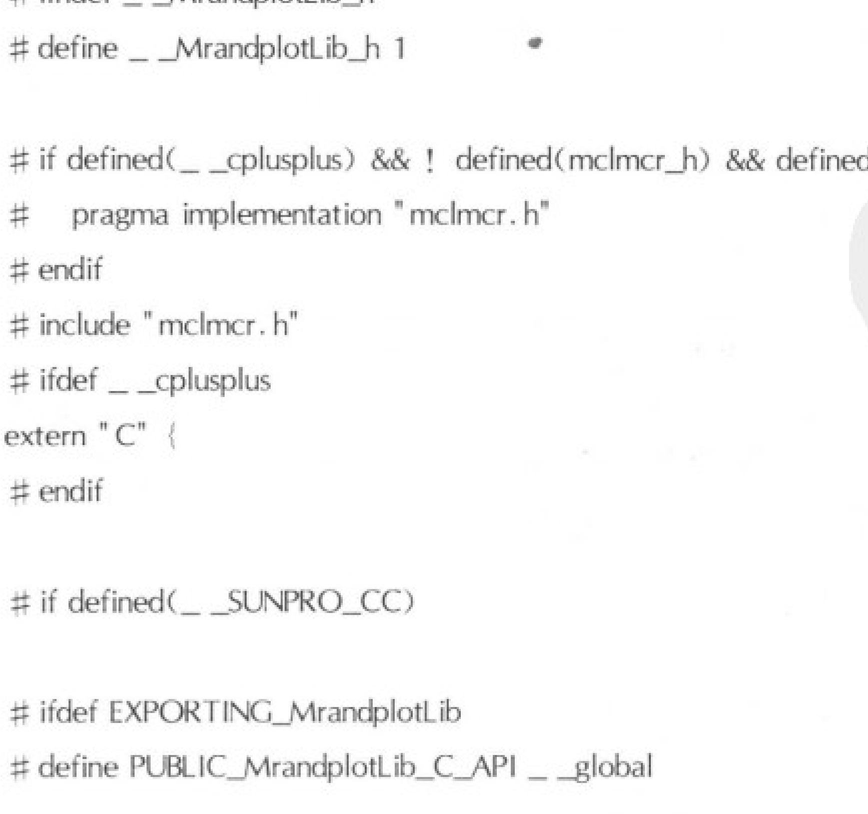
mcc -W lib:MrandplotLib mrandplot.m

便会生成如图 4-3 所示的文件。



图 4-3　编译 mrandplot.m 生成的文件

其中 mrandplot.h 的内容如下所示。



```
/*
 * Matlab Compiler: 4.6 (R2007a)
 * Date: Sun Jul 29 14:09:53 2007
 * Arguments: "-B" "macro_default" "-W" "lib:MrandplotLib" "mrandplot.m"
 */

#ifndef __MrandplotLib_h
#define __MrandplotLib_h 1

#if defined(__cplusplus) && ! defined(mclmcr_h) && defined(__linux__)
#    pragma implementation "mclmcr.h"
#endif
#include "mclmcr.h"
#ifdef __cplusplus
extern "C" {
#endif

#if defined(__SUNPRO_CC)

#ifdef EXPORTING_MrandplotLib
#define PUBLIC_MrandplotLib_C_API __global
```

```
#else
#define PUBLIC_MrandplotLib_C_API /* No import statement needed. */
#endif

#define LIB_MrandplotLib_C_API PUBLIC_MrandplotLib_C_API

#elif defined(_HPUX_SOURCE)

#ifdef EXPORTING_MrandplotLib
#define PUBLIC_MrandplotLib_C_API __declspec(dllexport)
#else
#define PUBLIC_MrandplotLib_C_API __declspec(dllimport)
#endif

#define LIB_MrandplotLib_C_API PUBLIC_MrandplotLib_C_API
#else
#define LIB_MrandplotLib_C_API
#endif

#ifndef LIB_MrandplotLib_C_API
#define LIB_MrandplotLib_C_API

#endif

extern LIB_MrandplotLib_C_API
bool MW_CALL_CONV
MrandplotLibInitializeWithHandlers(mclOutputHandlerFcn error_handler,
mclOutputHandlerFcn print_handler);

extern LIB_MrandplotLib_C_API
bool MW_CALL_CONV MrandplotLibInitialize(void);
extern LIB_MrandplotLib_C_API
void MW_CALL_CONV MrandplotLibTerminate(void);
extern LIB_MrandplotLib_C_API
bool MW_CALL_CONV mlxMrandplot(int nlhs,mxArray *plhs[],
                    int nrhs,mxArray *prhs[]);
extern LIB_MrandplotLib_C_API bool MW_CALL_CONV mlfMrandplot
                    (int nargout,mxArray ** noise,mxArray * n);

#ifdef __cplusplus
}
#endif
```

```
#endif
```

其中：

```
bool mlfMrandplot(int nargout,mxArray * * noise,mxArray * n)
```

便是编译 mrandplot. m 文件后生成的 mrandplot 函数的接口函数。在 C 语言函数中可以通过调用 mlfMrandplot 函数来达到调用 mrandplot 函数的目的。

2. 调用 mlfMrandplot 函数

调用 mlfMrandplot 函数的 C 语言主函数如下所示。

```
/* mrandplot_vc_window.c 文件 */
#include <windows.h>
#include "MrandplotLib.h"
#include "string.h"
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    mxArray * out=NULL;
    mxArray * in=NULL;
    int isOK;
    double * pOut=NULL;
    mwSize i=0;
    mwSize nNum;
    char * pBuff=NULL;
    mwSize n=0;
    mwSize nSum=0;

    pBuff=(char *)malloc(5000 * sizeof(char));
    memset(pBuff,0x00,5000 * sizeof(char));

    mclInitializeApplication(NULL,0);
    isOK=MrandplotLibInitialize();


    in=mxCreateDoubleMatrix(1,1,mxREAL);
    * mxGetPr(in)=20;

    isOK=mlfMrandplot(1,&out,in);



    pOut=mxGetPr(out);
    nNum=mxGetNumberOfElements(out);
```



```
    for(i=0;i<nNum;i++)
    {
        n=sprintf(pBuff+nSum," %f\n",pOut[i]);
        nSum=nSum+n;
    }

    MessageBox(NULL,pBuff,"MRANDDATA",MB_OK);

    //等待 Matlab 图形窗口
    mclWaitForFiguresToDie(NULL);
    free(pBuff);
    mxDestroyArray(out);
    mxDestroyArray(in);
    MrandplotLibTerminate();
    mclTerminateApplication();
    return 0;
}
```

这个程序有两点需要注意：

① Matlab 和 C/C++混合编程中经常要用到 mx -API 函数。本例中调用 mxCreateDoubleMatrix API 函数来构造 mlfMrandplot 函数的输入参数。

② 如果 Matlab 程序中存在图形窗口，则需要调用 mclWaitForFiguresToDie 函数等待图形窗口执行完毕以后再退出，否则会出现错误。

3. 采用 Visual C++ 6.0 编译

采用 Matlab mbuild 也可以编译上述程序(mrandplot_vc_window. c)，但是不能调试程序，因而最好采用 Visual C++ 6.0 进行程序编译，这样可以充分利用 VC 工具的强大功能调试程序。采用 Visual C++ 6.0 编译程序的步骤如下所述。

① 建立一个 Visual C++ 6.0 Win32 Application 空工程。

② 设置 Visual C++ 6.0 工程。通过选择 Project|Settings 菜单项打开的对话框中的 link|input 选项，将库文件 libemlrt. lib、libmex. lib、libut. lib、mclmcrrt. lib、libeng. lib、libmwlapack. lib、mclcom. lib、mclxlmain. lib、libfixedpoint. lib、libmwservices. lib、mclcommain. lib、libdflapack. lib、libmat. lib、libmx. lib、mclmcr. lib 加入到 Visual C++ 6.0 工程中(这些库文件在〈matlabroot〉\extern\lib\win32\microsoft 目录下)。

③ 将 mrandplot_vc_window. c、MrandplotLib_mcc_component_data. c、MrandplotLib. c、MrandplotLib. h 加入到工程中。

④ 编译后运行。

此程序利用了 Win32 程序设计的特点，采用 WinMain 函数而不是传统的控制台程序作为入口函数，因而可以去掉不太雅观的控制台窗口，但是需要读者有一些 Windows 程序设计的基础知识，这一技巧在 4.1.4 小节中进行详细讲解。mrandwin. exe 的运行结果如图 4－4 所示。

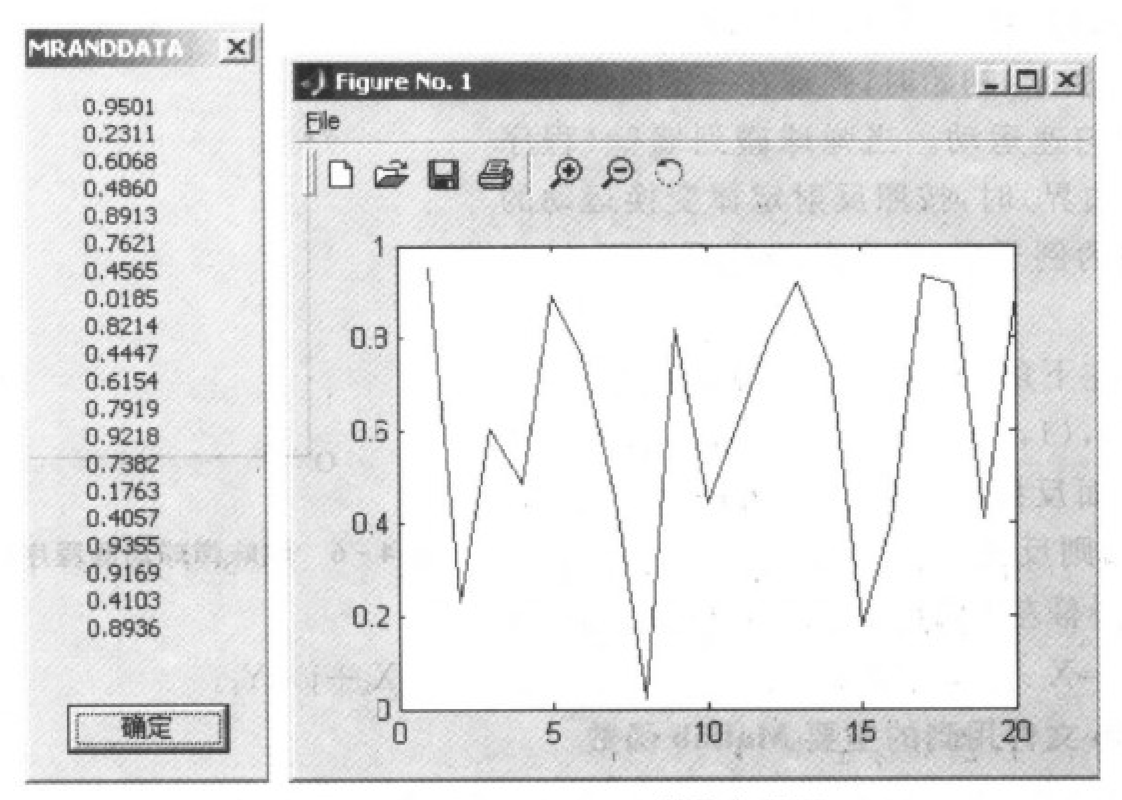


图 4-4　Mrandwin.exe 的执行结果

4.4　利用 Visual C++编译 M 文件并去掉控制台窗口

1. 采用 WinMain 代替 main

采用 Matlab GUI GUIDE 可以编辑和生成 Matlab 的图形界面程序，然后再通过采用 Visual C++ 编译生成的 *.m 文件，从而达到生成独立可执行程序的目的。

下面就用一个趣味弹球的动画程序（如图 4-5 所示）来说明这个思路的可行性。

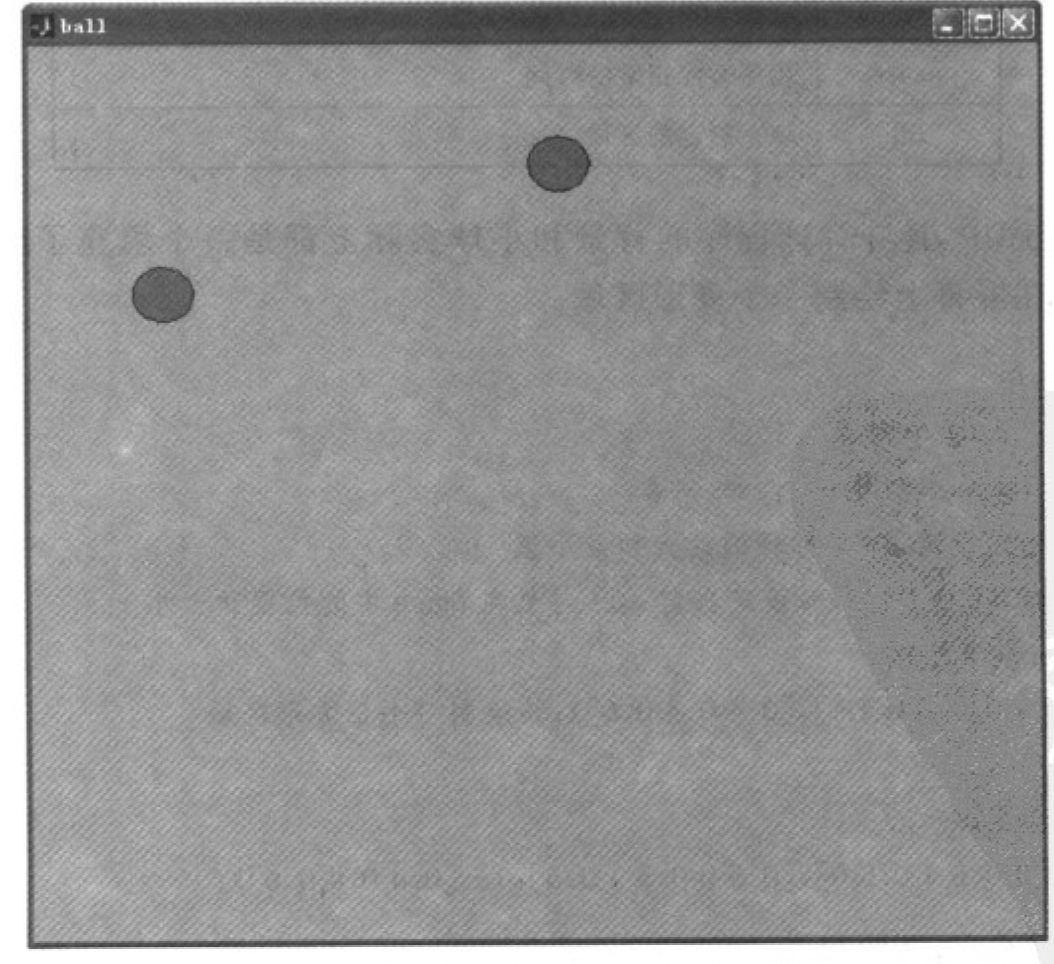


图 4-5　趣味弹球动画程序的运行结果

(1) 趣味弹球动画程序的运行机理

如图 4-5 所示，初始时，弹球在一定的位置按照一定的方向匀速运动。当弹球碰到墙壁(程序中图形界面的边界)时，按照反射定律变换运动的方向。在下面的例子中，设定的坐标系如图 4-6 所示。

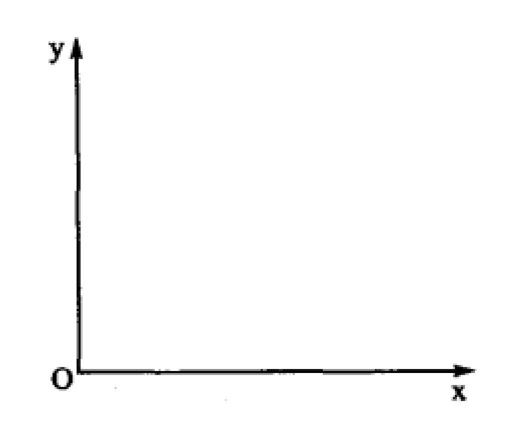


图 4-6 趣味弹球动画程序坐标系示意图

设定墙壁左下角为坐标原点 O，墙壁四个顶点的坐标为(0,0)，(1,0)，(1,1)，(0,1)，则当弹球在屏幕上下墙壁面反射时，假定入射方向的矢量为 $Z_i = X_j + j \times Y_i$，则反射方向的矢量为 $Z_i = X_i - j \times Y_i$；当弹球在屏幕左右墙壁面反射时，假定入射方向的矢量为 $Z_i = X_i + j \times Y_i$，则反射方向的矢量为 $Z_i = -X_i + j \times Y_i$。

(2) ball.m 文件用到的主要 Matlab 函数

下面简单介绍 ball.m 程序中用到的主要 Matlab 函数，如表 4-1 所列。

表 4-1 趣味弹球程序用到的主要 Matlab 函数列表

名　称	作　用
set	设置指定对象的制定属性
gcf	返回当前 figure 对象的 Matlab 句柄
gca	返回当前 axes 对象的 Matlab 句柄
hold on	保存当前 plot 和 axes 对象的各属性值，以便在同一幅图上显示多次绘图的结果
drawnow	通知 Matlab 强制刷新屏幕
pause	程序暂停指定的时间
fill	填充制定的区域

在 ball.m 程序中，趣味弹球程序的背景和小球实际上都是一个填充了颜色的区域，例如下面的语句就是在屏幕上绘制一个菱形区域。

```
%drawdiamond.m
figure;%创建一个 figure 对象
axes;%在当前 figure 中创建一个 axes 对象
axis([0 1 0 1]);%设定 XY 坐标轴的最大和最小值
set(gca,'position',[0 0 1 1]);%设定当前 axes 对象在 figure 中的位置和大小
axis off;%隐藏 xy 坐标轴
hrect=fill([0 1 1 0],[0 0 1 1],[0.4 0.4 0.6]);%绘制一个正方形区域
hold on;
%绘制一个菱形区域
hdiamond=fill([0 0.5 1 0.5 0],[0.5 0 0.5 1 0.5],1-[0.4 0.4 0.6]);
```

这段程序的执行结果如图 4-7 所示。

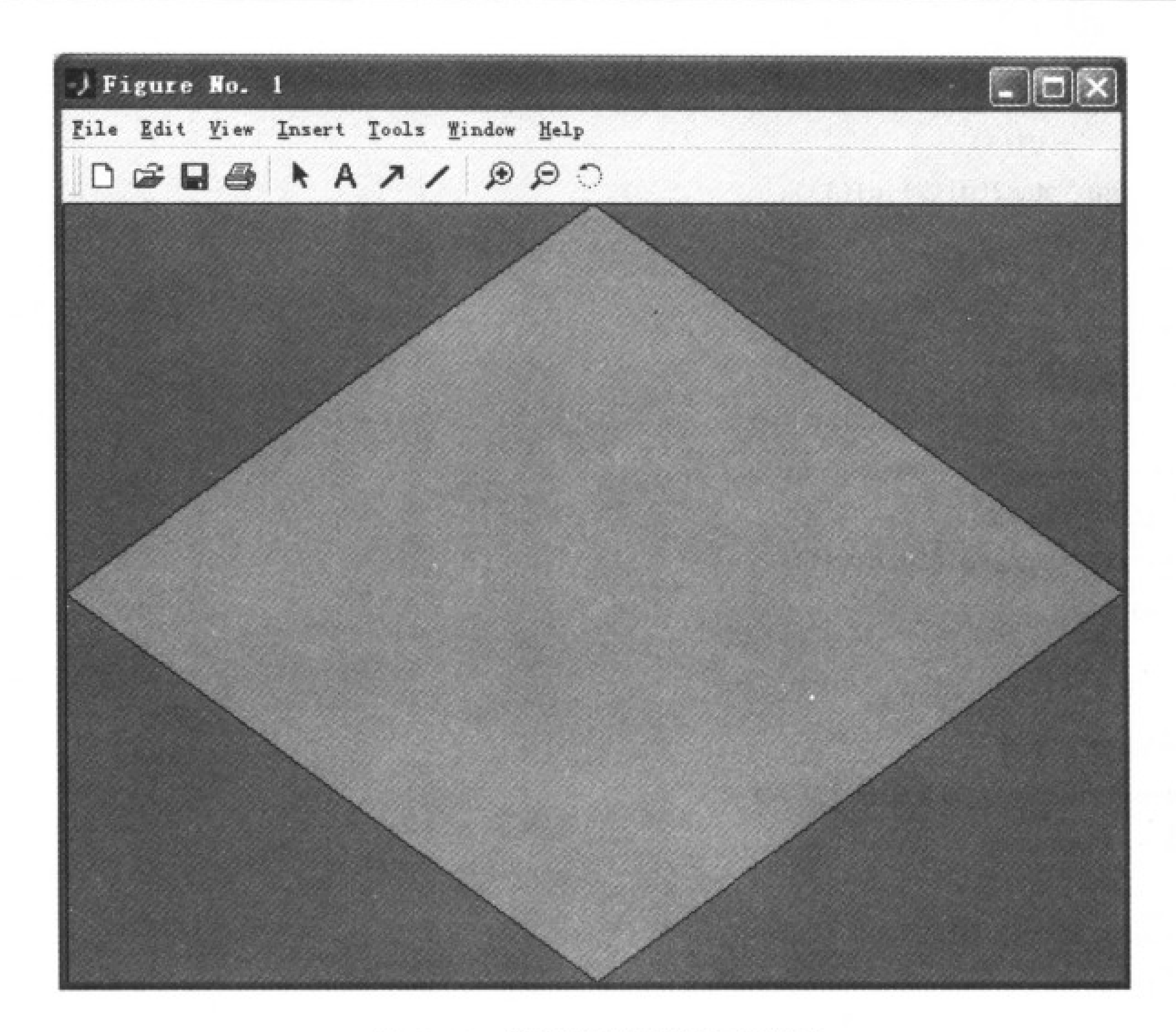


图 4-7　菱形填充测试程序结果

(3) 趣味弹球动画的处理程序

由于篇幅所限，这里只对本例需要的函数进行简单的讲解。如果读者对上述函数仍有疑问，还可以通过 help 函数得到更加详细的帮助。下面编写一个函数用于判断弹球运动的下一个位置的坐标和运动方向等参数。这里将其定义为 getnextball，该函数的实现代码如下所示。

```
function [newxdata, newydata, newposition, newdire] = ...
            getnextball(xdata, ydata, preposition, predire, dx, dy)
% function [newxdata, newydata, newposition, newdire] =
% getnextball(xdata, ydata, preposition, predire, dx, dy)
% 计算弹球运行的下一个位置
newxdata = xdata + dx * cos(predire);
newydata = ydata + dy * sin(predire);
newposition = preposition + [dx * cos(predire) dy * sin(predire)];
% 弹球碰到右壁面
if newposition(1) + 0.03 + 0.001 >= 1
    d1 = newposition - preposition;
    d1(1) = -d1(1);
    newdire = atan2(d1(2), d1(1));
    return;
end

% 弹球碰到左壁面
```

```
if newposition(1) - 0.03 - 0.001<=0
    d1 = newposition - preposition;
    d1(1) = - d1(1);
    newdire = atan2(d1(2),d1(1));
    return;
end

%弹球碰到上壁面
if newposition(2) - 0.03 - 0.001<=0
    d1 = newposition - preposition;
    d1(2) = - d1(2);
    newdire = atan2(d1(2),d1(1));
    return;
end

%弹球碰到下壁面
if newposition(2) + 0.03 + 0.001>=1
    d1 = newposition - preposition;
    d1(2) = - d1(2);
    newdire = atan2(d1(2),d1(1));
    return;
end
%没有碰到任何壁面
newdire = predire;
```

另外,在 Matlab GUI GUIDE 自动生成的函数 ball_OpeningFcn 中加入下面的初始化代码和趣味弹球的运行程序。

```
%初始化当前 figure 对象
set(gcf,'color',[0.5 0.7 0.3]);%
set(gcf,'doublebuffer','on');
%建立并初始化用于动画绘制的坐标轴
axes(handles.axes1);
set(gca,'color',1-[0.5 0.7 0.3]);
set(gca,'position',[[0 0 1 1]]);
axis([0 1 0 1]);
hold on;
%绘制趣味弹球动画的背景
handles.h1 = fill([0 1 1 0],[0 0 1 1],1-[0.5 0.7 0.3]);
handles.h2 = fill([0 1 1 0],[0 0 1 1],1-[0.5 0.7 0.3]);
%初始化两个弹球
handles.direction1 = 30/180 * 2 * pi + rand(1,1) * 30/180 * 2 * pi;%方向
handles.direction2 = 30/180 * 2 * pi + rand(1,1) * 30/180 * 2 * pi;
axis off
handles.go = 1;
handles.position1 = [0.5 0.5];%位置
```

```
handles.position2=[0.3 0.3];
handles.dx=0.01;
handles.dy=0.01;
handles.xdata1=0.03*cos(0:0.1:2*pi)+handles.position1(1);%弹球 1 的区域坐标
handles.ydata1=0.03*sin(0:0.1:2*pi)+handles.position1(2);
handles.xdata2=0.03*cos(0:0.1:2*pi)+handles.position2(1);%弹球 2 的区域坐标
handles.ydata2=0.03*sin(0:0.1:2*pi)+handles.position2(2);
while handles.go < 100
    %计算弹球运动的下一个位置
    [handles.xdata1,handles.ydata1,handles.position1,handles.direction1]=...
                getnextball(handles.xdata1,handles.ydata1,...
                    handles.position1,handles.direction1,handles.dx,...
                    handles.dy);
    [handles.xdata2,handles.ydata2,handles.position2,handles.direction2]=...
                getnextball(handles.xdata2,handles.ydata2,...
    handles.position2,handles.direction2,handles.dx,...
                            handles.dy);
    %更新弹球的位置
    set(handles.h1,'XData',handles.xdata1,'Ydata',handles.ydata1);
    set(handles.h2,'XData',handles.xdata2,'Ydata',handles.ydata2);
    drawnow;
    pause(0.03);%程序运行暂停 0.03s,用于控制弹球的运动速度
end
```

(4) 采用 Matlab mcc 命令将 ball.m 编译为独立可执行文件

采用命令 mcc -m ball.m 编译 ball.m,生成独立可执行文件 ball.exe 并执行。其执行结果如图 4-8 所示。由图 4-8 可以看出,由于有一个控制台程序的黑窗口,因而大大影响了程序执行后的效果。在下节中,通过采用 Visual C++编译 ball.m 文件,并采用将 main 函数替换为 WinMain 函数的做法达到将 Console(控制台)程序转换为 Win32 应用程序以及去掉控制体黑窗口的目的。

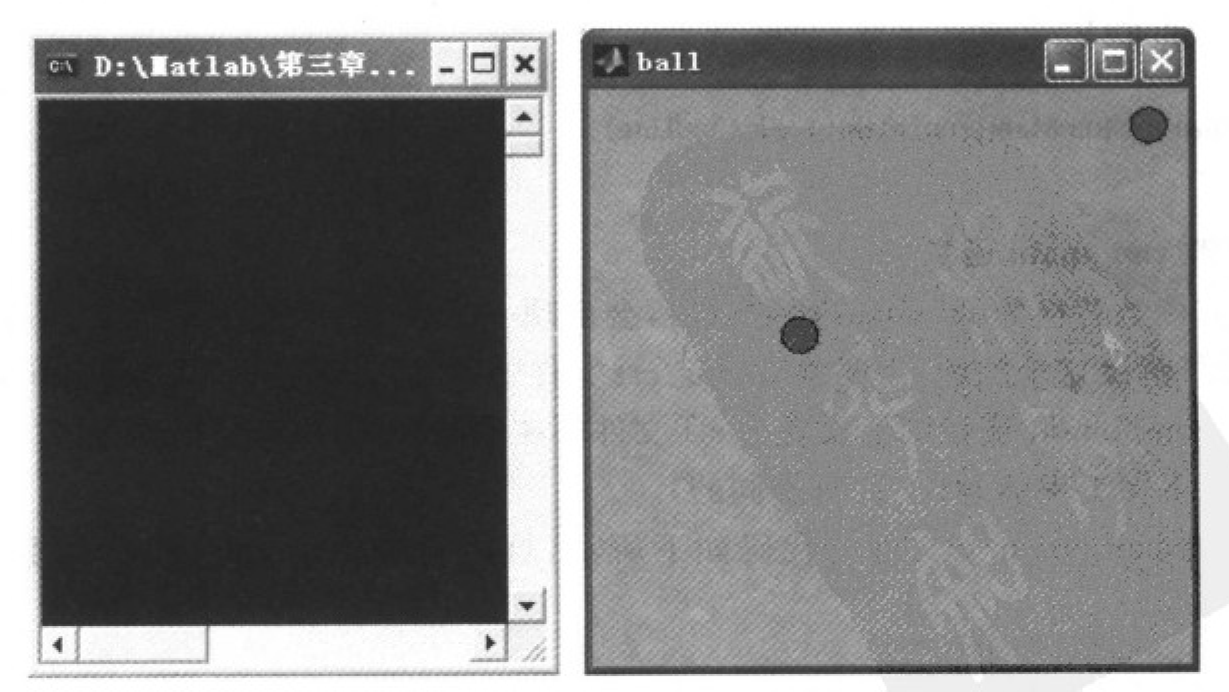


图 4-8　带控制台窗口的 ball 程序运行结果

(5) 采用 Matlab 编译其编译 ball. m 文件

采用 mcc -m ball. m 编译 ball. m 文件以后会生成 ball_main. c、ball_mcc_component_data. c 文件,注意 ball_main. c 文件中的 run_main 函数和 main 函数,通过修改这两个函数可以达到去掉控制台窗口的目的。

(6) 将 main 函数转换为 WinMain 函数

要去掉控制台窗口,需要将控制台类型的程序转换为 Win32 类型的程序;因为 Win32 程序的入口函数为 WinMain,因而需要将 main 函数转换为 WinMain 函数。WinMain 函数的输入输出参数与 main 函数不同,函数的主体部分与 main 函数类似。

```
/* 自动生成的 main 函数被隐藏 */
/* int main(int argc,const char * * argv)
{
    if (! mclInitializeApplication(
        __MCC_ball_component_data. runtime_options,
        __MCC_ball_component_data. runtime_option_count))
        return 0;

    return mclRunMain(run_main,argc,argv);
} */


/* * 下面这段程序经过修改和自动生成的 main 函数功能相同 * */
int __stdcall WinMain(HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine,
                      int nShowCmd)
{
    if (! mclInitializeApplication(
        __MCC_ball_component_data. runtime_options,
        __MCC_ball_component_data. runtime_option_count))
        return 0;

    return mclRunMain(run_main,1,&lpCmdLine);
}
```

(7) 修改 run_main 函数

当 main 函数替换为 winmain 函数以后,会出现一个问题,即程序不能自动获取 ctf 文件的路径。为了解决这个问题,需要手工设定 ctf 文件的路径,即将__MCC_ball_component_data. path_to_component 变量的值设置为 ctf 文件所在的路径(用户需要根据自己工程的情况进行改动,因为不同工程的 ctf 文件是不同的)。

修改后的 ball_main. c 文件的代码如下所示(这里只列出了 run_main 函数和 winmain 函数)。

```
int run_main(int argc,const char * * argv)
{
```

```
    int _retval;
    /* Generate and populate the path_to_component. */
    char path_to_component[(PATH_MAX * 2) + 1];
    separatePathName(argv[0],path_to_component,(PATH_MAX * 2) + 1);
    //__MCC_ball_component_data.path_to_component = path_to_component;
    __MCC_ball_component_data.path_to_component = "D:\\Matlab\\第 3 章\\采用 Winmain 代替
Main";
    if (! ballInitialize()) {
        return -1;
    }
    _retval = mclMain(_mcr_inst,argc,argv,"ball",1);
    if (_retval == 0 /* no error */) mclWaitForFiguresToDie(NULL);
    ballTerminate();
    mclTerminateApplication();
    return _retval;
}
```

(8) 采用 VC 将 ball_main.c 编译为独立可执行文件

当 ball_main.c 文件修改完成以后，就可以采用 VC 进行编译了，编译步骤如下所述。

① 首先通过 VC 建立一个 Win32 Application 类型的空工程，如图 4-9 所示。

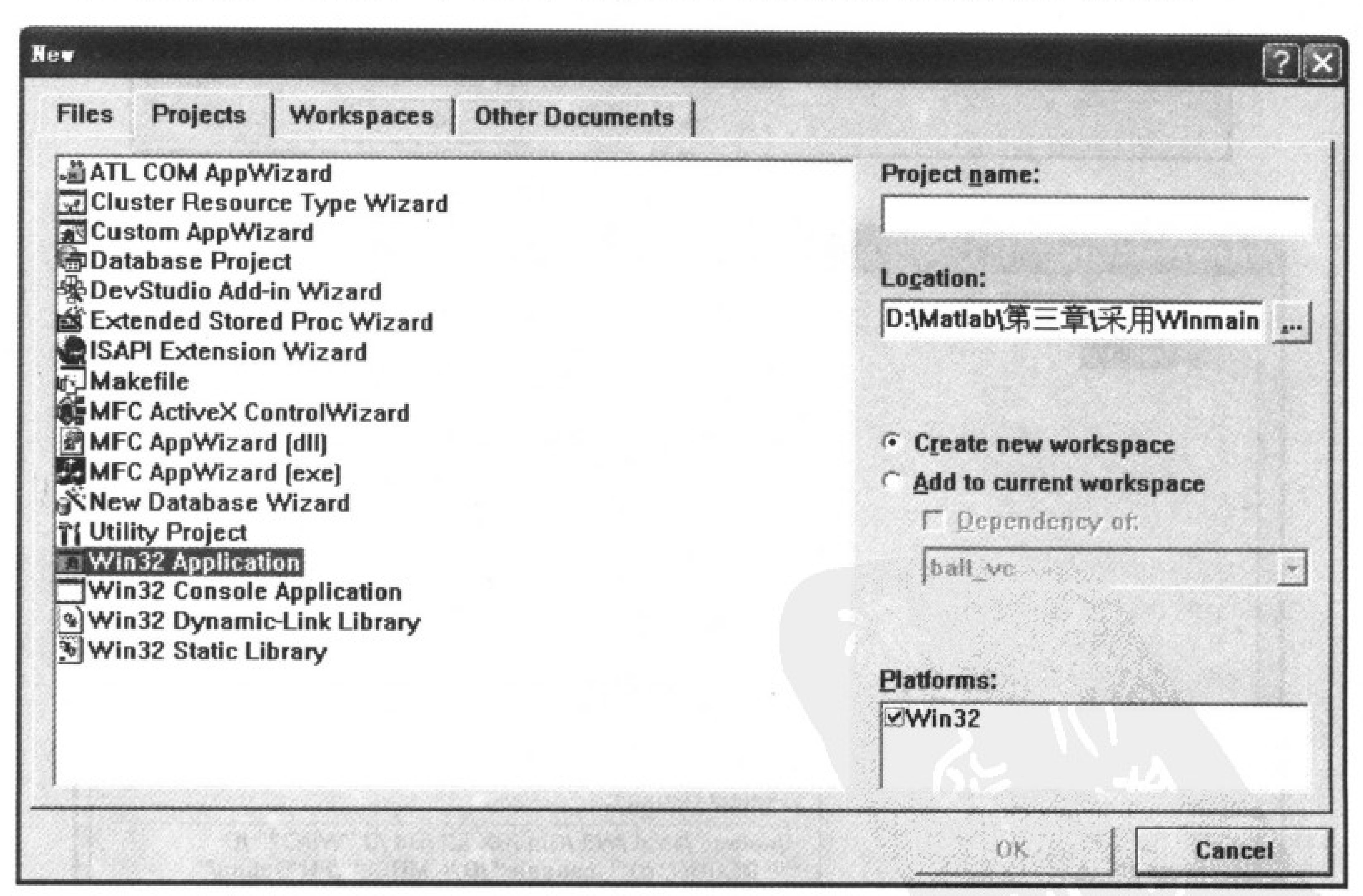


图 4-9　建立 Win32 Application 空工程

② 将 ball_main.c 文件和 ball_mcc_component_data.c 文件加入到新建立的 VC++工程中。

③ 通过 Project|Settings 菜单项更改 VC++工程的设置。首先通过 link 选项添加库文件：libemlrt. lib、libmex. lib、libut. lib、mclmcrrt. lib、libeng. lib、libmwlapack. lib、mclcom. lib、mclxlmain. lib、libfixedpoint. lib、libmwservices. lib、mclcommain. lib、libdflapack. lib、libmat. lib、libmx. lib、mclmcr. lib(如图 4-10 所示)。然后通过 C++|Precompiled Header 菜单项自动应用 stdafx. h 文件(如图 4-11 所示)。

图 4-10 添加库文件

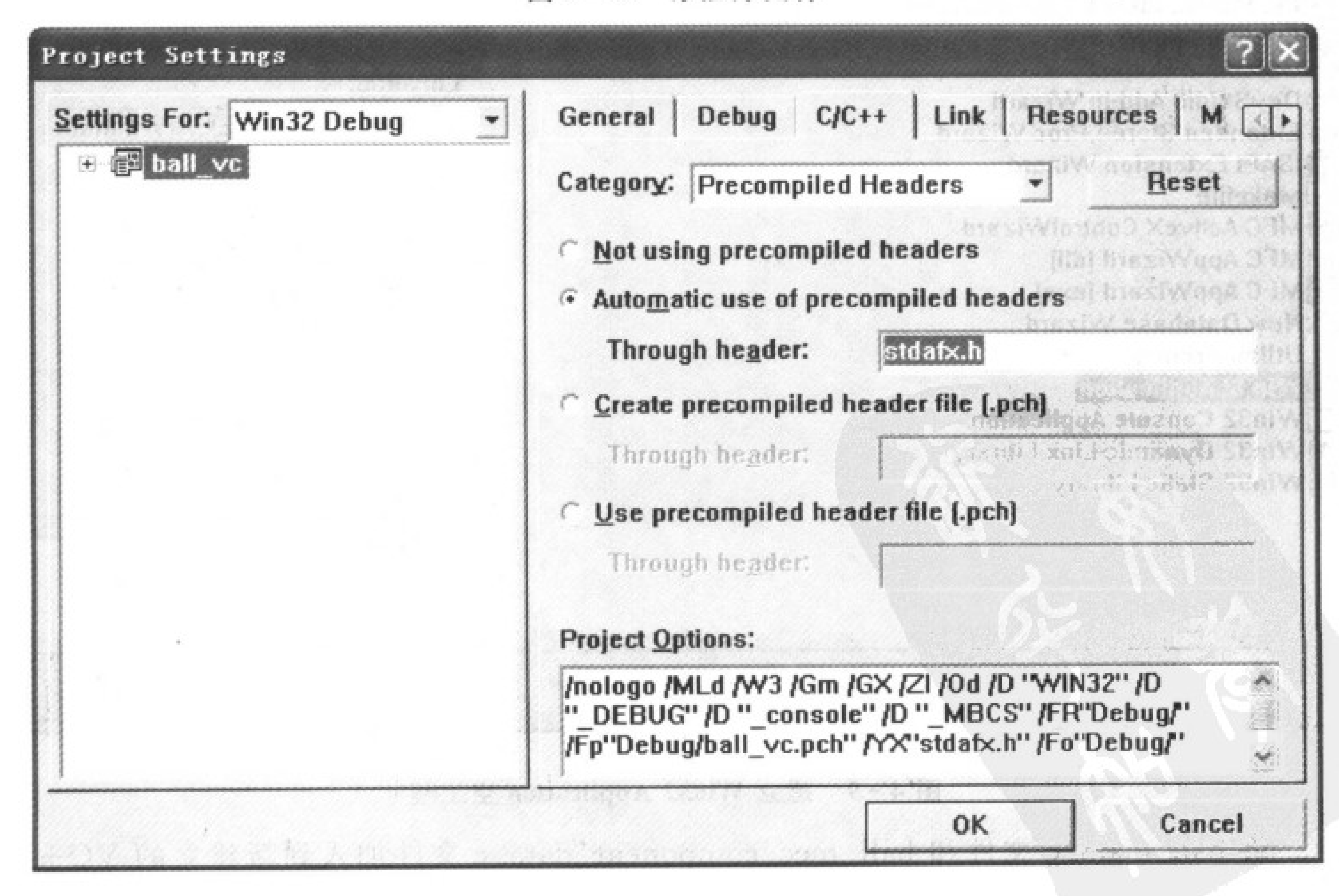


图 4-11 自动添加 stdafx. h 文件

④ 编译得到 ball.exe 可执行文件。最后运行 ball.exe 程序,此时生成的趣味弹球程序就不会再出现控制台窗口,而只剩下程序运行的窗口了。

为了方便读者对照和调试程序,将 ball.m 的代码全部列出,如下所示。

```
function varargout = ball(varargin)
% BALL M - file for ball.fig
%       BALL,by itself,creates a new BALL or raises the existing
%       singleton * .
%
%       H = BALL returns the handle to a new BALL or the handle to
%       the existing singleton * .
%
%       BALL('CALLBACK',hObject,eventData,handles,...) calls the local
%       function named CALLBACK in BALL.M with the given input arguments.
%
%       BALL('Property','Value',...) creates a new BALL or raises the
%       existing singleton * .  Starting from the left,property value pairs are
%       applied to the GUI before ball_OpeningFunction gets called.  An
%       unrecognized property name or invalid value makes property application
%       stop.  All inputs are passed to ball_OpeningFcn via varargin.
%
%       * See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE,GUIDATA,GUIHANDLES

% Edit the above text to modify the response to help ball

% Last Modified by GUIDE v2.5 30 - Sep - 2004 23:17:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename,...
                   'gui_Singleton',  gui_Singleton,...
                   'gui_OpeningFcn',@ball_OpeningFcn,...
                   'gui_OutputFcn',  @ball_OutputFcn,...
                   'gui_LayoutFcn',  [] ,...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
```

```
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ball is made visible.
function ball_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of Matlab
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ball (see VARARGIN)

% Choose default command line output for ball
handles.output = hObject;
%初始化当前 figure 对象
set(gcf,'color',[0.5 0.7 0.3]);%
set(gcf,'doublebuffer','on');
%建立并初始化用于动画绘制的坐标轴
axes(handles.axes1);
set(gca,'color',1-[0.5 0.7 0.3]);
set(gca,'position',[[0 0 1 1]]);
axis([0 1 0 1]);
hold on;
%绘制趣味弹球动画的背景
handles.h1 = fill([0 1 1 0],[0 0 1 1],1-[0.5 0.7 0.3]);
handles.h2 = fill([0 1 1 0],[0 0 1 1],1-[0.5 0.7 0.3]);
%初始化两个弹球
handles.direction1 = 30/180 * 2 * pi + rand(1,1) * 30/180 * 2 * pi;%方向
handles.direction2 = 30/180 * 2 * pi + rand(1,1) * 30/180 * 2 * pi;
axis off
handles.go = 1;
handles.position1 = [0.5 0.5];%位置
handles.position2 = [0.3 0.3];
handles.dx = 0.01;
handles.dy = 0.01;
handles.xdata1 = 0.03 * cos(0:0.1:2 * pi) + handles.position1(1);%弹球 1 的区域坐标
handles.ydata1 = 0.03 * sin(0:0.1:2 * pi) + handles.position1(2);
handles.xdata2 = 0.03 * cos(0:0.1:2 * pi) + handles.position2(1);%弹球 2 的区域坐标
handles.ydata2 = 0.03 * sin(0:0.1:2 * pi) + handles.position2(2);
while handles.go < 100
```

```
    %计算弹球的运动的下一个位置
    [handles.xdata1,handles.ydata1,handles.position1,handles.direction1] = ...
            getnextball(handles.xdata1,handles.ydata1,...
handles.position1,handles.direction1,handles.dx,...
handles.dy);
    [handles.xdata2,handles.ydata2,handles.position2,handles.direction2] = ...
            getnextball(handles.xdata2,handles.ydata2,...
handles.position2,handles.direction2,handles.dx,...
handles.dy);
    %更新弹球的位置
set(handles.h1,'XData',handles.xdata1,'Ydata',handles.ydata1);
    set(handles.h2,'XData',handles.xdata2,'Ydata',handles.ydata2);
    drawnow;
    pause(0.03);%程序运行暂停 0.03s,用于控制弹球的运动速度
end

% Update handles structure
guidata(hObject,handles);

% UIWAIT makes ball wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = ball_OutputFcn(hObject,eventdata,handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of Matlab
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function [newxdata,newydata,newposition,newdire] = ...
getnextball(xdata,ydata,preposition,predire,dx,dy)
% function [newxdata,newydata,newposition,newdire] =
%                      getnextball(xdata,ydata,preposition,predire,dx,dy)
%计算弹球运行的下一个位置
newxdata = xdata + dx * cos(predire);
newydata = ydata + dy * sin(predire);
newposition = preposition + [dx * cos(predire) dy * sin(predire)];
%弹球碰到右壁面
if newposition(1) + 0.03 + 0.001 >= 1
```

```
    d1 = newposition - preposition;
    d1(1) = -d1(1);
    newdire = atan2(d1(2),d1(1));
    return;
end

%弹球碰到左壁面
if newposition(1) - 0.03 - 0.001 <= 0
    d1 = newposition - preposition;
    d1(1) = -d1(1);
    newdire = atan2(d1(2),d1(1));
    return;
end

%弹球碰到上壁面
if newposition(2) - 0.03 - 0.001 <= 0
    d1 = newposition - preposition;
    d1(2) = -d1(2);
    newdire = atan2(d1(2),d1(1));
    return;
end

%弹球碰到下壁面
if newposition(2) + 0.03 + 0.001 >= 1
    d1 = newposition - preposition;
    d1(2) = -d1(2);
    newdire = atan2(d1(2),d1(1));
    return;
end
%没有碰到任何壁面
newdire = predire;
```

2. 去掉控制台窗口的另一种方法

除了可以采用 4.1.4 小节第 1 部分中将 main 函数改为 WinMain 函数的做法之外，还可以采用将 Matlab 编译完成的独立可执行文件直接运行，并通过 FindWindows 函数找到控制台窗口并将其隐藏。这样做得好处是可以随时显示控制台窗口中 Matlab 编译的程序的运行信息。本例中仍采用 plotsin.m 作为运行实例。

```
function [] = plotsin()
% function [] = plotsin()
x = -pi:0.01:pi;
y = sin(x);
plot(x,y,'+');
```

通过命令 mcc -m plotsin.m 将 plotsin.m 编译为 plotsin.exe，然后用 Visual C++ 6.0 建立 testconsole 对话框应用程序，如图 4-12 所示。其中四个按钮的 ID 号如表 4-2 所列。



图 4-12　testconsole 对话框程序

表 4-2　testconsole 按钮和 ID 对照表

ID	按钮名称
IDOK	绘图
IDC_HIDECONSOLE	隐藏控制台
IDC_SHOWCONSOLE	显示控制台
IDCANCEL	退出

选择 View|ClassWizard 菜单项实现上述按钮的鼠标单击消息映射函数，如图 4-13 所示。

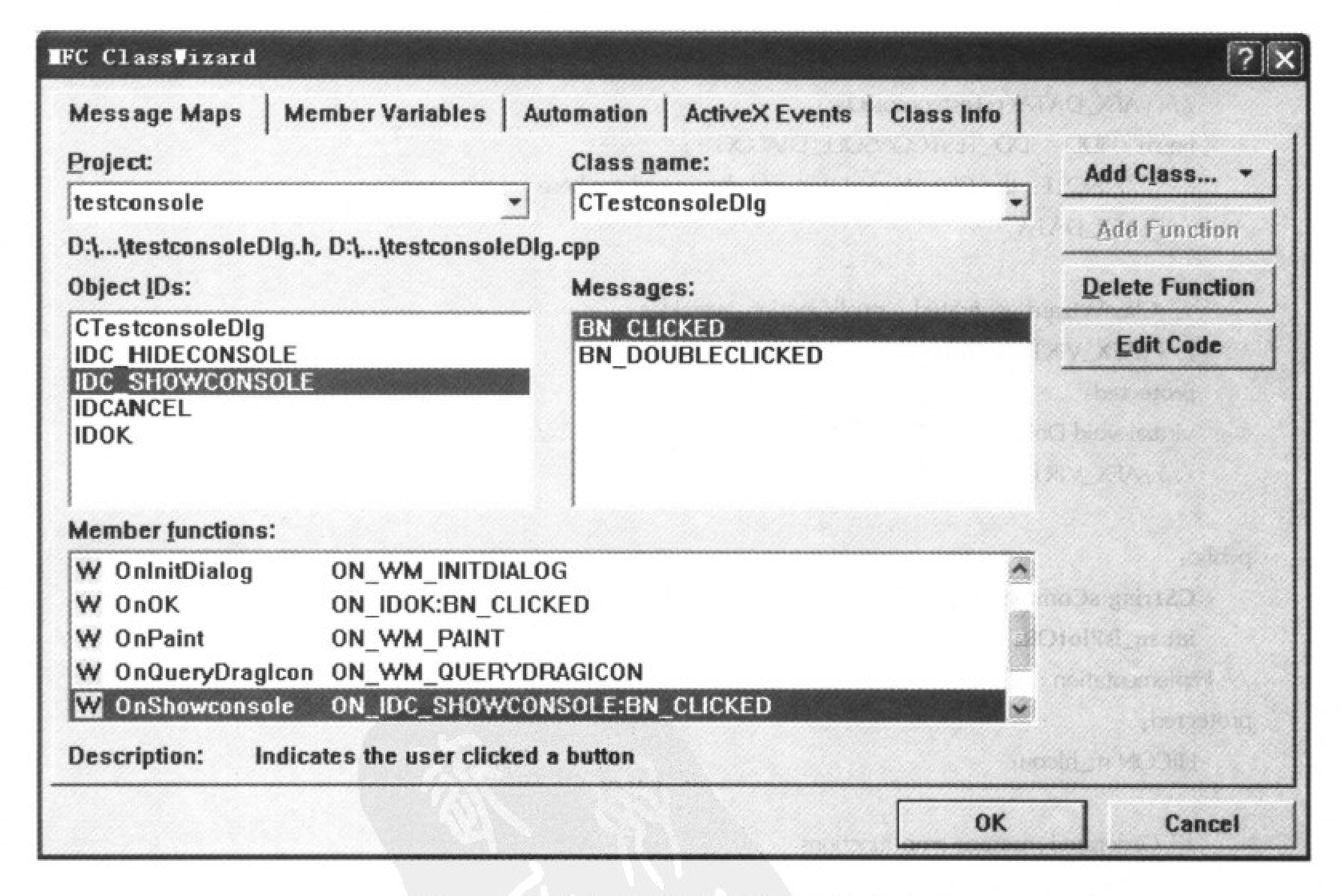


图 4-13　实现按钮的鼠标单击消息映射函数

下面给出实现本例的主要代码。

```
// testconsoleDlg.h : header file
//

#if ! defined(AFX_TESTCONSOLEDLG_H__A3CCF86E_2545_4E0B_8BDB_32299F4EAE13__INCLUDED_)
```

```
# define
AFX_TESTCONSOLEDLG_H__A3CCF86E_2545_4E0B_8BDB_32299F4EAE13__INCLUDED_

# if _MSC_VER > 1000
# pragma once
# endif
// _MSC_VER > 1000

////////////////////////////////////////////////////////////////////////////
// CTestconsoleDlg dialog

class CTestconsoleDlg : public CDialog
{
// Construction
public:
     CTestconsoleDlg(CWnd * pParent = NULL);      // standard constructor

// Dialog Data
     //{{AFX_DATA(CTestconsoleDlg)
     enum { IDD = IDD_TESTCONSOLE_DIALOG };
          // NOTE: the ClassWizard will add data members here
     //}}AFX_DATA

     // ClassWizard generated virtual function overrides
     //{{AFX_VIRTUAL(CTestconsoleDlg)
     protected:
     virtual void DoDataExchange(CDataExchange * pDX);      // DDX/DDV support
     //}}AFX_VIRTUAL

public:
     CString sCommandLine;
     int m_isPlotOk;
// Implementation
protected:
     HICON m_hIcon;

     // Generated message map functions
     //{{AFX_MSG(CTestconsoleDlg)
     virtual BOOL OnInitDialog();
     afx_msg void OnSysCommand(UINT nID,LPARAM lParam);
     afx_msg void OnPaint();
     afx_msg HCURSOR OnQueryDragIcon();
     virtual void OnOK();
     afx_msg void OnHideconsole();
     afx_msg void OnTimer(UINT nIDEvent);
```

```
    afx_msg void OnShowconsole( );
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // ! defined(AFX_TESTCONSOLEDLG_H__A3CCF86E_2545_4E0B_8BDB_32299F4EAE13__INCLUDED
_)


// testconsoleDlg.cpp : implementation file
//

#include "stdafx.h"
#include "testconsole.h"
#include "testconsoleDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About
char cCommandLine[MAX_PATH];

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL
```

```
// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange * pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestconsoleDlg dialog

CTestconsoleDlg::CTestconsoleDlg(CWnd * pParent /* =NULL */)
    : CDialog(CTestconsoleDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CTestconsoleDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    char curDir[MAX_PATH];
    ::GetCurrentDirectory(MAX_PATH,curDir);
    sCommandLine.Format("%s",curDir);
    sCommandLine += "\\plotsin.exe";
    m_isPlotOk=0;
}

void CTestconsoleDlg::DoDataExchange(CDataExchange * pDX)
```

```
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CTestconsoleDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTestconsoleDlg,CDialog)
    //{{AFX_MSG_MAP(CTestconsoleDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_HIDECONSOLE,OnHideconsole)
    ON_WM_TIMER()
    ON_BN_CLICKED(IDC_SHOWCONSOLE,OnShowconsole)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestconsoleDlg message handlers

BOOL CTestconsoleDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING,IDM_ABOUTBOX,strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
```

```
    SetIcon(m_hIcon,TRUE);                // Set big icon
    SetIcon(m_hIcon,FALSE);           // Set small icon

    // TODO: Add extra initialization here

    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CTestconsoleDlg::OnSysCommand(UINT nID,LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID,lParam);
    }
}

// If you add a minimize button to your dialog,you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CTestconsoleDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,(WPARAM) dc.GetSafeHdc(),0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x,y,m_hIcon);
    }
    else
```

```
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTestconsoleDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CTestconsoleDlg::OnOK()
{
    // TODO: Add extra validation here

    PROCESS_INFORMATION pi;
    STARTUPINFO si={sizeof(si)};

    //启动子进程
    ::strcpy(cCommandLine,sCommandLine);
    CreateProcess(NULL,cCommandLine,NULL,NULL,FALSE,0,NULL,NULL,&si,&pi);
    m_isPlotOk=1;
    SetTimer(0,100,NULL);
    //CDialog::OnOK();
}

void CTestconsoleDlg::OnHideconsole()
{
    // TODO: Add your control notification handler code here

    HWND m_hwndPlot=::FindWindow(NULL,sCommandLine);
    if(m_hwndPlot!=NULL)
    {
        ::ShowWindow(m_hwndPlot,SW_HIDE);
    }
}

void CTestconsoleDlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    if(m_isPlotOk)
    {
            HWND m_hwndPlot=::FindWindow(NULL,sCommandLine);
            if(m_hwndPlot!=NULL)
```

```
            {
                ::ShowWindow(m_hwndPlot,SW_HIDE);
            }
            m_isPlotOk = 0;
    }
    CDialog::OnTimer(nIDEvent);
}

void CTestconsoleDlg::OnShowconsole()
{
    // TODO: Add your control notification handler code here
    HWND m_hwndPlot = ::FindWindow(NULL,sCommandLine);
    if(m_hwndPlot! =NULL)
    {
        ::ShowWindow(m_hwndPlot,SW_SHOW);
    }
}
```

执行 testconsole 程序的结果如图 4－14 所示。

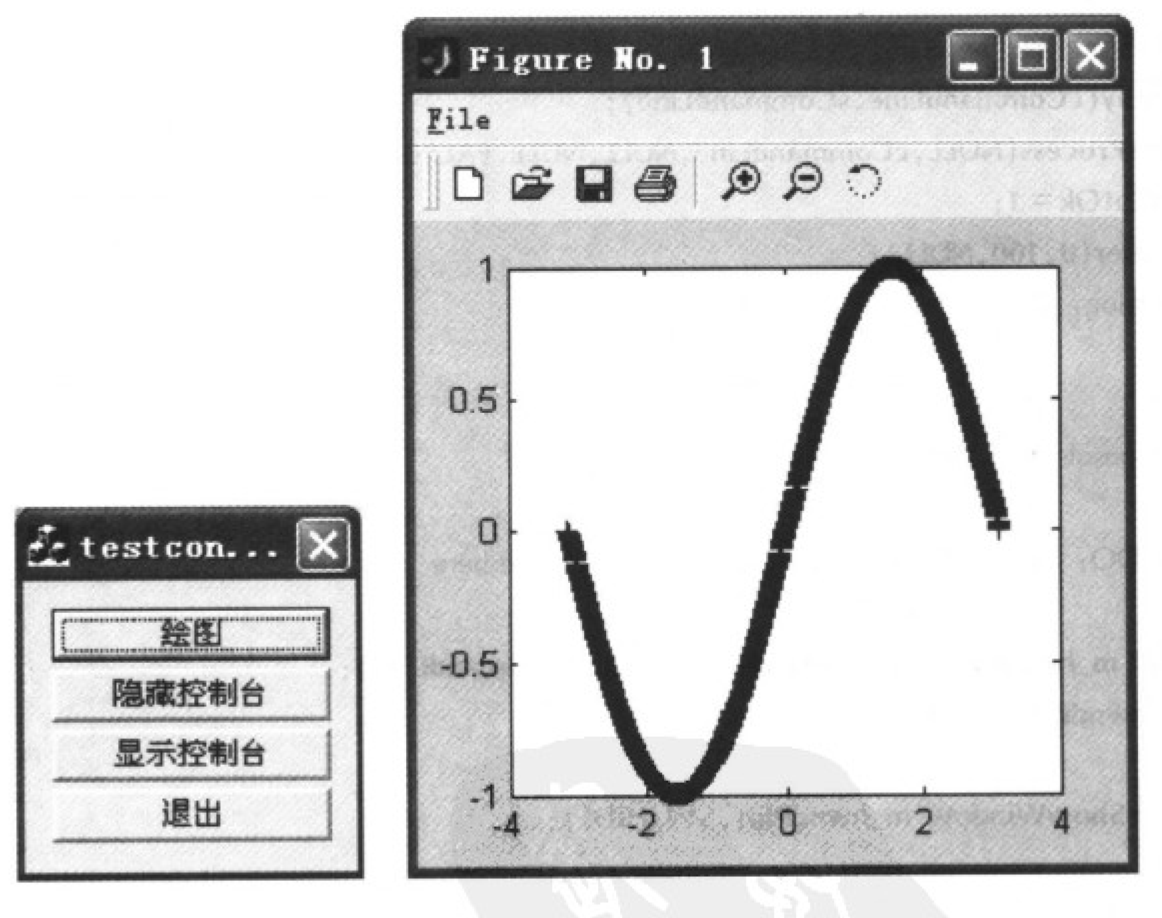


图 4－14　testconsole 程序的运行结果

3. 将输出改到 Windows 窗口上

Matlab 编译后的程序都运行在命令行模式下，其输出信息也都显示在 DOS 命令行窗口中。通过采用适当的方法，虽然可以将 DOS 命令行窗口隐掉，但是有些有用的程序输出信息却看不到了。本节就是通过一个实例来说明如何将编译后的 Matlab 程序的输出信息输出到一个 Windows 窗口上。在 codeguru 上找到一个关于将控制台程序的输入和输出定位到一个 CEditView 窗口的实例，其网址是：http://www. codeguru. com/misc/RedirectOutput-

ToPipe. shtml。

下面就是从 CeditView 派生的 CshellView 的源代码。

```
/*************shellview.h****************/
#if ! defined(AFX_SHELLVIEW_H__10EC4EE4_E283_11D2_9B23_004005649FB5__INCLUDED_)
#define AFX_SHELLVIEW_H__10EC4EE4_E283_11D2_9B23_004005649FB5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// ShellView.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CShellView view

class CShellView : public CEditView
{
protected:
    CShellView();            // protected constructor used by dynamic creation
    DECLARE_DYNCREATE(CShellView)

private:
    HANDLE hChildStdinRd,hChildStdinWr,hChildStdinWrDup,
        hChildStdoutRd,hChildStdoutWr,hChildStdoutRdDup,
        hSaveStdin,hSaveStdout;
    CWinThread *    m_pReadThread;
    DWORD dwProcessId;

    LOGFONT m_lf;
    CFont m_defFont;
// Attributes
public:

// Operations
public:
    void GetUserInput( CString& input );
    void AddTexts( LPCTSTR string );

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CShellView)
    protected:
    virtual void OnDraw(CDC * pDC);      // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
```

```
// Implementation
protected:
    virtual ~CShellView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

    // Generated message map functions
protected:
    BOOL CreateShellRedirect();
    //{{AFX_MSG(CShellView)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();
    afx_msg void OnChar(UINT nChar,UINT nRepCnt,UINT nFlags);
    afx_msg void OnKeyDown(UINT nChar,UINT nRepCnt,UINT nFlags);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
private:
    void AddTexts( TCHAR ch );
    void WriteToPipe(LPCTSTR line);
    BOOL CreateChildProcess(DWORD& dwProcessId);

    static UINT ReadPipeThreadProc( LPVOID pParam );
private:
    int GetSelLength();
    int GetCurrentPosition();
    void MoveToEnd();
    int m_nLength;
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif
// !defined(AFX_SHELLVIEW_H__10EC4EE4_E283_11D2_9B23_004005649FB5__INCLUDED_)



/**********shellview.cpp**************/
// ShellView.cpp : implementation file
//

#include "stdafx.h"
#include "ShellView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CShellView

IMPLEMENT_DYNCREATE(CShellView,CEditView)

CShellView::CShellView()
{
    dwProcessId=DWORD(-1);
    m_pReadThread=NULL;

    //    Initialize LOGFONT structure
    memset(&m_lf,0,sizeof(m_lf));
    m_lf.lfWeight=FW_NORMAL;
    m_lf.lfCharSet=GB2312_CHARSET;//ANSI_CHARSET;
    m_lf.lfOutPrecision=OUT_DEFAULT_PRECIS;
    m_lf.lfClipPrecision=CLIP_DEFAULT_PRECIS;
    m_lf.lfQuality=DEFAULT_QUALITY;
    m_lf.lfPitchAndFamily=DEFAULT_PITCH | FF_DONTCARE;
    lstrcpy(m_lf.lfFaceName,_T("FixedSys"));
}

CShellView::~CShellView()
{
}

BEGIN_MESSAGE_MAP(CShellView,CEditView)
    //{{AFX_MSG_MAP(CShellView)
    ON_WM_CREATE()
    ON_WM_DESTROY()
    ON_WM_CHAR()
    ON_WM_KEYDOWN()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CShellView drawing

void CShellView::OnDraw(CDC* pDC)
{
    CDocument* pDoc=GetDocument();
```

```
    // TODO: add draw code here
}

/////////////////////////////////////////////////////////////////////////////
// CShellView diagnostics

#ifdef _DEBUG
void CShellView::AssertValid() const
{
    CEditView::AssertValid();
}

void CShellView::Dump(CDumpContext& dc) const
{
    CEditView::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CShellView message handlers

int CShellView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CEditView::OnCreate(lpCreateStruct) == -1)
        return -1;

//  GetEditCtrl().SetReadOnly( TRUE );
    DWORD dwStyle = GetWindowLong( GetEditCtrl().GetSafeHwnd(),GWL_STYLE );
    if( dwStyle )
    {
        dwStyle |= DS_LOCALEDIT;
        SetWindowLong( GetEditCtrl().GetSafeHwnd(),GWL_STYLE,dwStyle );
    }

    if( m_defFont.CreateFontIndirect( &m_lf ) )
        GetEditCtrl().SetFont( &m_defFont );
    if( ! CreateShellRedirect() )
        return -1;

    return 0;
}

#define BUFSIZE 4096

BOOL CShellView::CreateShellRedirect()
{
    SECURITY_ATTRIBUTES saAttr;
```

```
BOOL fSuccess;

// Set the bInheritHandle flag so pipe handles are inherited.
saAttr.nLength = sizeof(SECURITY_ATTRIBUTES);
saAttr.bInheritHandle = TRUE;
saAttr.lpSecurityDescriptor = NULL;

// The steps for redirecting child process's STDOUT:
//        1. Save current STDOUT, to be restored later.
//        2. Create anonymous pipe to be STDOUT for child process.
//        3. Set STDOUT of the parent process to be write handle to
//           the pipe, so it is inherited by the child process.
//        4. Create a noninheritable duplicate of the read handle and
//           close the inheritable read handle.

// Save the handle to the current STDOUT.
hSaveStdout = GetStdHandle(STD_OUTPUT_HANDLE);

// Create a pipe for the child process's STDOUT.
if( ! CreatePipe( &hChildStdoutRd, &hChildStdoutWr, &saAttr, 0) )
{
     TRACE0( _T("Stdout pipe creation failed\n") );
     return FALSE;
}

// Set a write handle to the pipe to be STDOUT.
if( ! SetStdHandle(STD_OUTPUT_HANDLE, hChildStdoutWr) )
{
     TRACE0( _T("Redirecting STDOUT failed\n") );
     return FALSE;
}

// Create noninheritable read handle and close the inheritable read handle.
fSuccess = DuplicateHandle( GetCurrentProcess(), hChildStdoutRd,
     GetCurrentProcess(),   &hChildStdoutRdDup ,
     0,   FALSE,
     DUPLICATE_SAME_ACCESS );
if( ! fSuccess )
{
     TRACE0( _T("DuplicateHandle failed\n") );
     return FALSE;
}
CloseHandle( hChildStdoutRd );

// The steps for redirecting child process's STDIN:
//        1.   Save current STDIN, to be restored later.
//        2.   Create anonymous pipe to be STDIN for child process.
```

```
//      3.  Set STDIN of the parent to be the read handle to the
//          pipe,so it is inherited by the child process.
//      4.  Create a noninheritable duplicate of the write handle,
//          and close the inheritable write handle.

// Save the handle to the current STDIN.
hSaveStdin = GetStdHandle(STD_INPUT_HANDLE);

// Create a pipe for the child process's STDIN.
if( ! CreatePipe(&hChildStdinRd,&hChildStdinWr,&saAttr,0) )
{
    TRACE0( _T("Stdin pipe creation failed\n") );
    return FALSE;
}
// Set a read handle to the pipe to be STDIN.
if( ! SetStdHandle(STD_INPUT_HANDLE,hChildStdinRd) )
{
    TRACE0( _T("Redirecting Stdin failed\n") );
    return FALSE;
}
// Duplicate the write handle to the pipe so it is not inherited.
fSuccess = DuplicateHandle(GetCurrentProcess(),hChildStdinWr,
    GetCurrentProcess(),&hChildStdinWrDup,
    0,FALSE,                    // not inherited
    DUPLICATE_SAME_ACCESS );
if( ! fSuccess )
{
    TRACE0( _T("DuplicateHandle failed\n") );
    return FALSE;
}
CloseHandle(hChildStdinWr);

// Now create the child process.
if( ! CreateChildProcess(dwProcessId) )
{
    TRACE0( _T("CreateChildProcess failed\n") );
    return FALSE;
}
// After process creation,restore the saved STDIN and STDOUT.
if( ! SetStdHandle(STD_INPUT_HANDLE,hSaveStdin) )
{
    TRACE0( _T("Re-redirecting Stdin failed\n") );
    return FALSE;
}
if( ! SetStdHandle(STD_OUTPUT_HANDLE,hSaveStdout) )
{
    TRACE0( _T("Re-redirecting Stdout failed\n") );
```

```
        return FALSE;
    }
    m_pReadThread = AfxBeginThread( (AFX_THREADPROC)ReadPipeThreadProc,(LPVOID)this );
    if( ! m_pReadThread )
    {
        TRACE0( _T("Cannot start read - redirect thread! \n") );
        return FALSE;
    }
    return TRUE;
}

BOOL CShellView::CreateChildProcess(DWORD& dwProcessId)
{
    PROCESS_INFORMATION piProcInfo;
    STARTUPINFO siStartInfo;

    // Set up members of STARTUPINFO structure.
    ZeroMemory( &siStartInfo,sizeof(STARTUPINFO) );
    siStartInfo.cb = sizeof(STARTUPINFO);

    siStartInfo.dwFlags = STARTF_USESTDHANDLES;
    siStartInfo.hStdInput = hChildStdinRd;
    siStartInfo.hStdOutput = hChildStdoutWr;
    siStartInfo.hStdError = hChildStdoutWr;

    TCHAR shellCmd[_MAX_PATH];
    if( ! GetEnvironmentVariable(_T("ComSpec"),shellCmd,_MAX_PATH) )
          return FALSE;
#ifdef _UNICODE
    _tcscat( shellCmd,_T(" /U") );
#else
    _tcscat( shellCmd,_T(" /A") );
#endif
    // Create the child process.
    BOOL ret = CreateProcess
                    (
                        NULL,
                        shellCmd,         // applicatin name
                        NULL,             // process security attributes
                        NULL,             // primary thread security attributes
                        TRUE,             // handles are inherited
                        DETACHED_PROCESS, // creation flags
                        NULL,             // use parent's environment
                        NULL,             // use parent's current directory
                        &siStartInfo,   // STARTUPINFO pointer
                        &piProcInfo
                    );  // receives PROCESS_INFORMATION
```

```
    if( ret )
    {
        dwProcessId = piProcInfo.dwProcessId;
    }
    return ret;
}

void CShellView::WriteToPipe( LPCTSTR line )
{
    DWORD dwWritten;

    WriteFile( hChildStdinWrDup,line,_tcslen(line) * sizeof(TCHAR),
            &dwWritten,NULL );
}

UINT CShellView::ReadPipeThreadProc( LPVOID pParam )
{
    DWORD dwRead;
    TCHAR chBuf[BUFSIZE];
    CShellView *  pView = (CShellView * )pParam;

    TRACE0( _T("ReadPipe Thread begin run\n") );
    for( ;; )
    {
        if( ! ReadFile( pView->hChildStdoutRdDup,chBuf,
            BUFSIZE,&dwRead,NULL) || dwRead == 0)
            break;
        chBuf[dwRead/sizeof(TCHAR)] = _T('\0');
        pView->AddTexts( chBuf );
        pView->m_nLength = pView->GetEditCtrl().SendMessage( WM_GETTEXTLENGTH );
    }
    CloseHandle( pView  ->hChildStdinRd);
    CloseHandle( pView  ->hChildStdoutWr);
    CloseHandle( pView  ->hChildStdinWrDup );
    CloseHandle( pView  ->hChildStdoutRdDup );
    pView->m_pReadThread = NULL;
    pView->dwProcessId = DWORD( - 1);
    pView->PostMessage( WM_CLOSE );
    return 1;
}


void CShellView::OnDestroy()
{
    if( dwProcessId!  = DWORD( - 1) )
```

```
    {
        HANDLE hProcess = OpenProcess( PROCESS_ALL_ACCESS,FALSE,dwProcessId );
        if( hProcess )
        {
            TerminateProcess( hProcess,0 );
            CloseHandle( hProcess );
        }
    }
    if( m_pReadThread )
    {
        TerminateThread( m_pReadThread->m_hThread,0 );
        delete m_pReadThread;
    }
    CEditView::OnDestroy();
}

void CShellView::AddTexts(LPCTSTR string)
{
    MoveToEnd();
    GetEditCtrl().ReplaceSel( string );
}

void CShellView::AddTexts(TCHAR ch)
{
    TCHAR string[2];
    string[0] = ch;
    string[1] = _T('\0');
    AddTexts( (LPCTSTR)string );
}

void CShellView::OnChar(UINT nChar,UINT nRepCnt,UINT nFlags)
{
    int nPos = GetCurrentPosition();
    if( nChar == 8 && nPos <= m_nLength )
        return;
    if( nPos<m_nLength )
        MoveToEnd();
    CEditView::OnChar(nChar,nRepCnt,nFlags);
    if( nChar == 13 )
    {
        CString input;
        GetUserInput(input);
        WriteToPipe( input );
```

```
    }
}

void CShellView::OnKeyDown(UINT nChar,UINT nRepCnt,UINT nFlags)
{
    if( nChar == VK_DELETE )
    {
        if( GetCurrentPosition()<m_nLength )
            return;
    }
    CEditView::OnKeyDown(nChar,nRepCnt,nFlags);
}

void CShellView::MoveToEnd()
{
    int nLen = GetEditCtrl().SendMessage( WM_GETTEXTLENGTH );
    GetEditCtrl().SetSel( nLen,nLen );
}

int CShellView::GetCurrentPosition()
{
    GetEditCtrl().SetSel( -1, -1);
    int nstart,nstop;
    GetEditCtrl().GetSel(nstart,nstop);
    return nstart;
}

void CShellView::GetUserInput(CString &input)
{
    int where = GetCurrentPosition();
    HLOCAL hBuffer = GetEditCtrl().GetHandle();
    if( hBuffer )
    {
        LPCTSTR szBuffer = (LPCTSTR)LocalLock(hBuffer);
        if( szBuffer )
        {
            input = CString( szBuffer + m_nLength,(where - m_nLength) );
            LocalUnlock( hBuffer );
        }
    }
}

int CShellView::GetSelLength()
```

```
{
    int nstart,nstop;
    GetEditCtrl().GetSel(nstart,nstop);
    return (nstart - nstop);
}

BOOL CShellView::PreCreateWindow(CREATESTRUCT& cs)
{
    return CEditView::PreCreateWindow(cs);
}
```

下面就在 CShellView 的基础上实现将 Matlab 程序的输出重定向到 Windows 窗口上。该例的实现步骤如下所述。

① 首先建立一个 Visual C++ 6.0 的单文档工程，工程名为 testredirectoutput，然后将 shellview.cpp 和 shellview.h 加入到新建的工程中来。

② 修改 testredirectoutputApp.cpp 文件中的 InitInstance 函数，将：

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CTestredirectoutputDoc),
    RUNTIME_CLASS(CMainFrame),          // main SDI frame window
    RUNTIME_CLASS(CTestredirectoutputView));
AddDocTemplate(pDocTemplate);
```

修改为：

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CTestredirectoutputDoc),
    RUNTIME_CLASS(CMainFrame),          // main SDI frame window
    RUNTIME_CLASS(CShellView));
AddDocTemplate(pDocTemplate);
```

这样做的目的是将文档模板默认的 CtestredirectoutputView 类改变为所需要的 CshellView 类。

③ 从工程中删除 testredirectoutputView.h 和 testredirectoutputView.h.cpp 文件，重新编译工程，如果出现引用 testredirectoutputView.h 文件的错误，直接将＃include "testredirectoutputView.h"删除即可。

④ 修改 shellview.cpp 的 CreateChildProcess 函数，如下所述。

```
// Create the child process.
BOOL ret = CreateProcess
                (
```

```
            NULL,
            shellCmd,         // applicatin name
            NULL,             // process security attributes
            NULL,             // primary thread security attributes
            TRUE,             // handles are inherited
            DETACHED_PROCESS,// creation flags
            NULL,             // use parent's environment
            NULL,             // use parent's current directory
            &siStartInfo,   // STARTUPINFO pointer
            &piProcInfo
        );  // receives PROCESS_INFORMATION
```

其中,将 shellCmd 改变为自己要运行的控制台程序 exe 文件的路径,在本例中设为"plotrand\\plotrand. exe",即 plotrand. exe 相对 testredirectoutput 工程的路径。如果指定路径中的可执行文件不存在,则会出现如图 4-15 所示的错误提示。



图 4-15　控制台可执行文件路径无效时存在的信息

⑤ 重新编译工程,然后运行程序,就会出现如图 4-16 所示的运行结果。从运行结果中可以看出,通过 CshellView 类,即可实现将 Matlab 编译后的控制台程序的执行结果输出到 Windows 窗口上的功能。

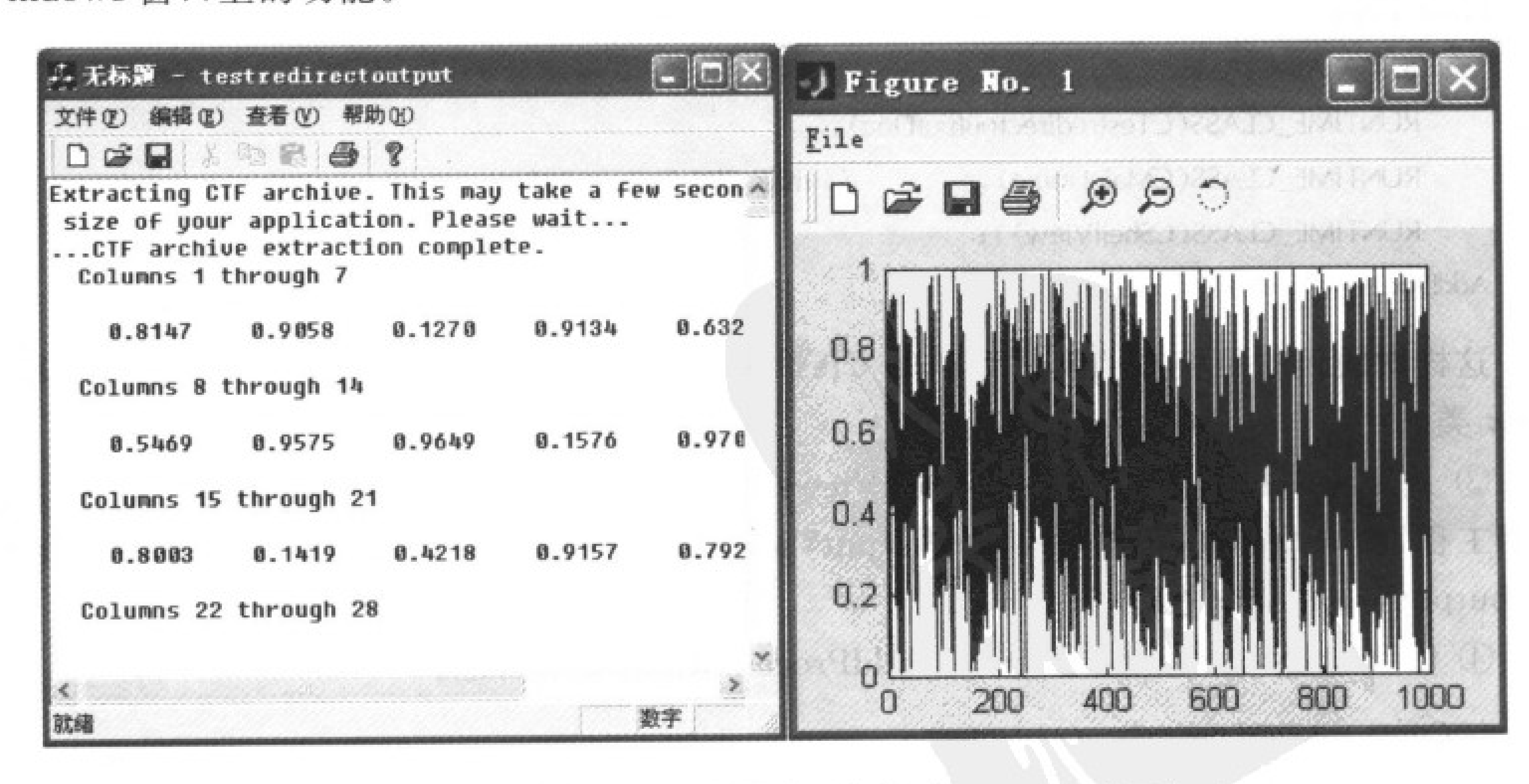


图 4-16　将 Matlab 程序输出重定向到 Windows 窗口上

第 5 章　Visual C++调用 Matlab 程序

5.1　在 Visual C++中调用 Matlab 引擎

Matlab 允许用户通过 Matlab 引擎调用 Matlab 的函数，即将 Matlab 当作应用程序进行数据计算的引擎。Matlab 提供了一系列操作 Matlab 引擎的 API 函数，通过这些 API 函数，用户可以充分发挥 Matlab 进行矩阵计算的优势，这样，应用程序的计算功能交给 Matlab 引擎完成，而界面部分则可以采用 VC++来实现。这便为应用程序的实现提供了很大的灵活性。

5.1.1　API 函数介绍

1. int engClose(Engine * ep);

此函数用于退出 Matlab 引擎。

2. int engEvalString(Engine * ep,const char * string);

此函数用于使 Matlab 引擎执行字符串 string 中的表达式。

3. mxArray * engGetVariable(Engine * ep,const char * name);

此函数用于从 Matlab 引擎工作空间中复制名字为 name 的变量。

4. int engGetVisible(Engine * ep,bool * value);

此函数用于判断 Matlab 引擎工作窗口是否可见。

5. Engine * engOpen(const char * startcmd);

此函数用于启动一个 Matlab 引擎，在 Windows 操作环境下 startcmd 参数必须为 NULL。

6. Engine * engOpenSingleUse(const char * startcmd,void * dcom,int * retstatus);

此函数用于启动一个只允许用户使用的 Matlab 引擎，在 Windows 系统中 startcmd 和 dcom 参数始终为 NULL，retstatus 返回 engOpenSingleUse 函数的执行状态。

7. int engOutputBuffer(Engine * ep,char * p,int n);

此函数用于设置 Matlab 引擎的输出内存，存储 engEvalString 函数后的输出结果。其中 n 表示设置的输出内存可以存放的字符个数，如果输出结果的字符串大于 n，则只存储前 n 个字符。

8. int engPutVariable(Engine * ep,const char * name,const mxArray * mp);

此函数用于向 Matlab 引擎工作空间中写入一个 Matlab 阵列变量，其中 name 为在 Matlab 引擎工作空间中写入的变量名字。

9. int engSetVisible(Engine * ep,bool value);

此函数用于设置 Matlab 引擎工作窗口是否可见的属性：如果 value=true，则 Matlab 引

擎窗口可见；如果 value=false，则 Matlab 引擎窗口不可见。

5.1.2　Visual C++调用 Matlab 引擎的实例

下面的实例展示了如何在 Visual C++ 6.0 MFC 工程中调用 Matlab 引擎。

首先创建一个单文档的 Visual C++ 6.0 工程 matlabenginetest，其中 view 类的基类选为 CFormView。

添加到 CMatlabenginetestView 中的界面元素及其 ID 如表 5-1 所列。

表 5-1　matlabenginetest CmatlabenginetestView 的控件列表

ID	控件类型	界面元素
IDC_StartEngine	Button	启动 Engine
IDC_DrawSinc	Button	计算并绘制 sinc 图像
IDC_CloseEngine	Button	关闭 Engine
IDC_HIDEENGINECHECK	Check button	隐藏 Matlab 引擎窗口
IDC_CMDEDIT	Edit	命令输入编辑框
IDC_EVALUATESTRING	Button	执行用户输入的命令
IDC_OUTPUTEDIT	Edit	Matlab 引擎结果显示编辑框

matlabenginetest 通过 engEvalString 函数调用 Matlab 引擎执行相应的命令计算并显示 sinc 函数曲线，通过函数 engOutputBuffer 将 m_outbuff 设为 Matlab 引擎输出结果存储的缓冲区。需要注意的是，由于本工程调用 Matlab 引擎的 API 函数，因而需要在工程设置中加入静态链接库 libeng.lib。下面给出了 matlabenginetest CMatlabenginetestView 实现的源代码和其运行结果(如图 5-1 所示)。

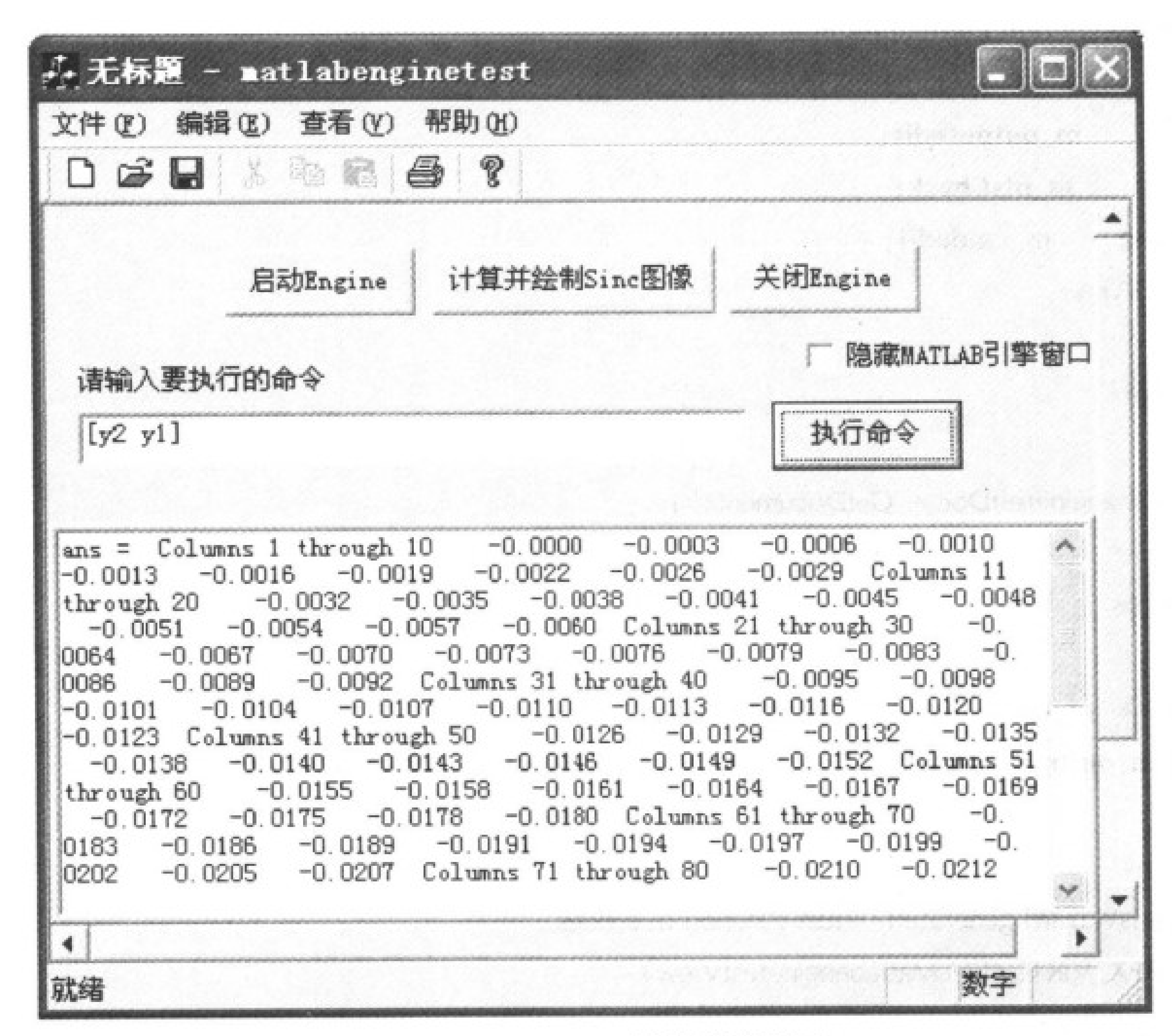
```
// matlabenginetestView.h : interface of the CMatlabenginetestView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MatlabENGINETESTVIEW_H__2BA448E9_DCB7_4CAB_8DE3_46FED317C086__INCLUDED_)
#define
AFX_MatlabENGINETESTVIEW_H__2BA448E9_DCB7_4CAB_8DE3_46FED317C086__INCLUDED_

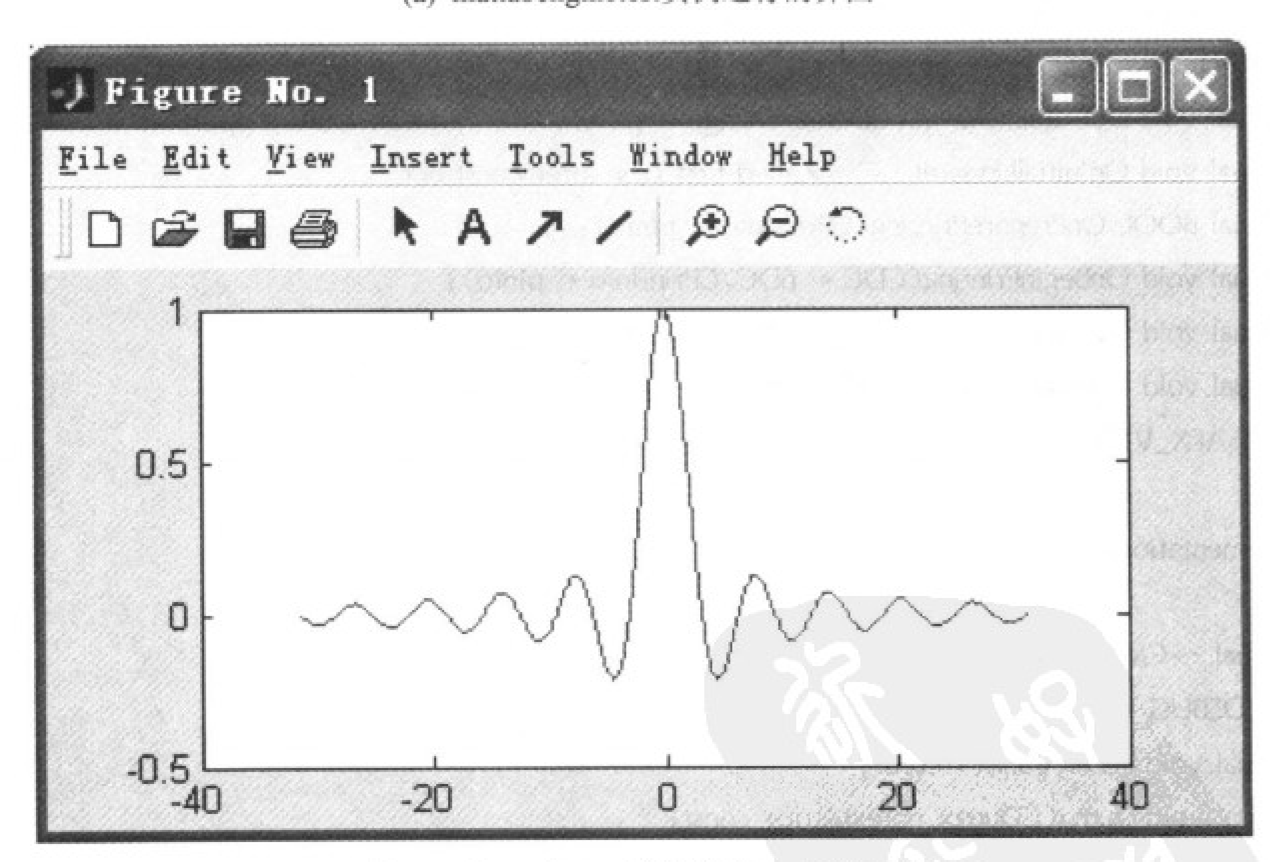
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "engine.h"
#define _MAX_BUFF_CHAR_NUM 2000

class CMatlabenginetestView : public CFormView
    {
protected: // create from serialization only
```



(a) matlabenginetest实例运行的界面



(b) matlabenginetest实例绘制sinc函数图像

图 5-1　matlabenginetest 实例

```
    CMatlabenginetestView();
    DECLARE_DYNCREATE(CMatlabenginetestView)

public:
```

```
    //{{AFX_DATA(CMatlabenginetestView)
    enum { IDD = IDD_MatlabENGINETEST_FORM };
    CEdit       m_outputedit;
    BOOL        m_nIsCheck;
    CString     m_cmdedit;
//}}AFX_DATA

// Attributes
public:
    CMatlabenginetestDoc * GetDocument();

// Operations
public:
    Engine * m_ep;//Matlab 引擎结构体
    char m_outbuff[_MAX_BUFF_CHAR_NUM];//Matlab 引擎的输出 buff

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMatlabenginetestView)
    public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual void DoDataExchange(CDataExchange * pDX);    // DDX/DDV support
    virtual void OnInitialUpdate(); // called first time after construct
    virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
    virtual void OnBeginPrinting(CDC * pDC,CPrintInfo * pInfo);
    virtual void OnEndPrinting(CDC * pDC,CPrintInfo * pInfo);
    virtual void OnPrint(CDC * pDC,CPrintInfo * pInfo);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMatlabenginetestView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CMatlabenginetestView)
```

```
    afx_msg void OnCloseEngine();
    afx_msg void OnDrawSinc();
    afx_msg void OnStartEngine();
    afx_msg void OnPaint();
    afx_msg void OnHideenginecheck();
    afx_msg void OnEvaluatestring();
    afx_msg void OnSize(UINT nType,int cx,int cy);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG   // debug version in matlabenginetestView.cpp
inline CMatlabenginetestDoc * CMatlabenginetestView::GetDocument()
   { return (CMatlabenginetestDoc *)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif
//! defined(AFX_MatlabENGINETESTVIEW_H__2BA448E9_DCB7_4CAB_8DE3_46FED317C086__INCLUDED_)


// matlabenginetestView.cpp : implementation of the CMatlabenginetestView class
//

#include "stdafx.h"
#include "matlabenginetest.h"

#include "matlabenginetestDoc.h"
#include "matlabenginetestView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMatlabenginetestView
```

```
IMPLEMENT_DYNCREATE(CMatlabenginetestView,CFormView)

BEGIN_MESSAGE_MAP(CMatlabenginetestView,CFormView)
    //{{AFX_MSG_MAP(CMatlabenginetestView)
    ON_BN_CLICKED(IDC_CloseEngine,OnCloseEngine)
    ON_BN_CLICKED(IDC_DrawSinc,OnDrawSinc)
    ON_BN_CLICKED(IDC_StartEngine,OnStartEngine)
    ON_WM_PAINT()
    ON_BN_CLICKED(IDC_HIDEENGINECHECK,OnHideenginecheck)
    ON_BN_CLICKED(IDC_EVALUATESTRING,OnEvaluatestring)
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT,CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT,CFormView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,CFormView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMatlabenginetestView construction/destruction

CMatlabenginetestView::CMatlabenginetestView()
    : CFormView(CMatlabenginetestView::IDD)
{
    //{{AFX_DATA_INIT(CMatlabenginetestView)
    m_nIsCheck = FALSE;
    m_cmdedit = _T("");
    //}}AFX_DATA_INIT
    // TODO: add construction code here
    m_ep = NULL;
    memset(m_outbuff,0,_MAX_BUFF_CHAR_NUM * sizeof(char));
}

CMatlabenginetestView::~CMatlabenginetestView()
{
    if(m_ep! =NULL)
    {
        engClose(m_ep);
    }
}

void CMatlabenginetestView::DoDataExchange(CDataExchange * pDX)
{
```

```
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMatlabenginetestView)
    DDX_Control(pDX,IDC_OUTPUTEDIT,m_outputedit);
    DDX_Check(pDX,IDC_HIDEENGINECHECK,m_nIsCheck);
    DDX_Text(pDX,IDC_CMDEDIT,m_cmdedit);
    //}}AFX_DATA_MAP
}

BOOL CMatlabenginetestView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //   the CREATESTRUCT cs

    return CFormView::PreCreateWindow(cs);
}

void CMatlabenginetestView::OnInitialUpdate()
{
    CFormView::OnInitialUpdate();
    GetParentFrame()->RecalcLayout();
    ResizeParentToFit();
    m_outputedit.SetWindowText("");
}

/////////////////////////////////////////////////////////////////////////////
// CMatlabenginetestView printing

BOOL CMatlabenginetestView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CMatlabenginetestView::OnBeginPrinting(CDC* /*pDC*/,CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CMatlabenginetestView::OnEndPrinting(CDC* /*pDC*/,CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}
```

```
void CMatlabenginetestView::OnPrint(CDC* pDC,CPrintInfo* /* pInfo */)
{
    // TODO: add customized printing code here
}

/////////////////////////////////////////////////////////////////////////////
// CMatlabenginetestView diagnostics

#ifdef _DEBUG
void CMatlabenginetestView::AssertValid() const
{
    CFormView::AssertValid();
}

void CMatlabenginetestView::Dump(CDumpContext& dc) const
{
    CFormView::Dump(dc);
}

CMatlabenginetestDoc* CMatlabenginetestView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMatlabenginetestDoc)));
    return (CMatlabenginetestDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMatlabenginetestView message handlers

void CMatlabenginetestView::OnCloseEngine()
{
    // TODO: Add your control notification handler code here
    engClose(m_ep);
    m_ep = NULL;
}

void CMatlabenginetestView::OnDrawSinc()
{
    // TODO: Add your control notification handler code here
    if(m_ep! = NULL)
    {
        //计算 sinc 函数并绘制 sinc 函数的图像
        engEvalString(m_ep,"x1 = 0.01:0.01:10 * pi");
```

```
        engEvalString(m_ep,"x2 = -10 * pi:0.01:-0.01");
        engEvalString(m_ep,"y1 = sin(x1)./x1");
        engEvalString(m_ep,"y2 = sin(x2)./x2");
        engEvalString(m_ep,"plot([x2 x1],[y2 y1])");
        //产生一个错误，从而验证 Matlab 引擎的输出已经被捕获
        engEvalString(m_ep,"1/0");
        m_outputedit.SetWindowText(m_outbuff);
    }
    else
    {
        AfxMessageBox("请启动 Matlab 引擎",MB_OK,NULL);
    }
}

void CMatlabenginetestView::OnStartEngine()
{
    // TODO: Add your control notification handler code here
    //打开 Matlab 引擎
    m_ep = engOpen(NULL);
    engOutputBuffer(m_ep,m_outbuff,_MAX_BUFF_CHAR_NUM);
}

void CMatlabenginetestView::OnPaint()
{
    CPaintDC dc(this); // device context for painting

    // TODO: Add your message handler code here
    // Do not call CFormView::OnPaint() for painting messages
}

void CMatlabenginetestView::OnHideenginecheck()
{
    // TODO: Add your control notification handler code here
    //改变 Matlab 引擎窗口的显示状态
    if(m_ep == NULL) return;
    UpdateData(true);
    engSetVisible(m_ep,! m_nIsCheck);
}

void CMatlabenginetestView::OnEvaluatestring()
{
    // TODO: Add your control notification handler code here
    //执行用户输入的命令
```

```
        UpdateData(true);
        LPSTR pstr = m_cmdedit.GetBuffer(m_cmdedit.GetLength());
        if(m_ep! = NULL)
        {
            engEvalString(m_ep,(const char * )pstr);
            m_outputedit.SetWindowText(m_outbuff);
        }
        else
        {
            AfxMessageBox("请启动 Matlab 引擎",MB_OK,NULL);
        }
}

void CMatlabenginetestView::OnSize(UINT nType,int cx,int cy)
{
        CFormView::OnSize(nType,cx,cy);

        // TODO: Add your message handler code here
        //父窗口改变的时候,
        //改变 Matlab 引擎输出结果 Edit 窗口的大小
        if(::IsWindow(m_outputedit.GetSafeHwnd()))
        {
            m_outputedit.MoveWindow(0,140,cx,cy - 140,true);
        }
}
```

5.2 Visual C++中调用 Matlab *.m 函数编译后的动态链接库

通过 Matlab 编译器的 mcc 命令,可以将 Matlab *.m 文件直接编译为动态链接库供 Visual C++调用。以 plotsinc 函数为例,采用命令 mcc -B csharedlib:libsincplot sincplot.m 将其编译为动态链接库,生成的文件列表如图 5-2 所示。

```
%sincplot.m 文件
function [y] = sincplot(n);
x1 = 0.01:0.01:n * pi;
y1 = sin(x1)./x1;
x2 = -n * pi:0.01:-0.01;
y2 = sin(x2)./x2;
y = [y2 y1];
plot([x2 x1],y);
```

接着,创建一个 Visual C++单文档的工程 testsincplot_dll,然后通过选择 Project|Settings

菜单项打开的对话框中的 link|input 选项，将库文件 libemlrt. lib、libmex. lib、libut. lib、mclmcrrt. lib、libeng. lib、libmwlapack. lib、mclcom. lib、mclxlmain. lib、libfixedpoint. lib、libmwservices. lib、mclcommain. lib、libdflapack. lib、libmat. lib、libmx. lib、mclmcr. lib、libsincplot. lib 加入到 Visual C++ 6.0 工程中(这些库文件在〈matlabroot〉\extern\lib\win32\microsoft目录下)，其中 libsincplot. lib 为上述步骤中由 Matlab编译器生成的文件之一。

将 libsincplot. dll 复制到 Windows 系统目录(system 或者 system32)或者 testsincplot_dll 工程的 debug 目录下，将 libsincplot. h、libsincplot. lib 和 libsincplot. ctf 文件复制到 testsincplot_dll 工程的源文件目录下，并将 libsincplot. h 文件添加到工程中。

为工具条添加一个按钮，其 ID 命名为 ID_PLOTSINC，如图 5-3 所示。

通过 ClassWizard 在 CTestsincplot_dllView中为其添加消息响应。

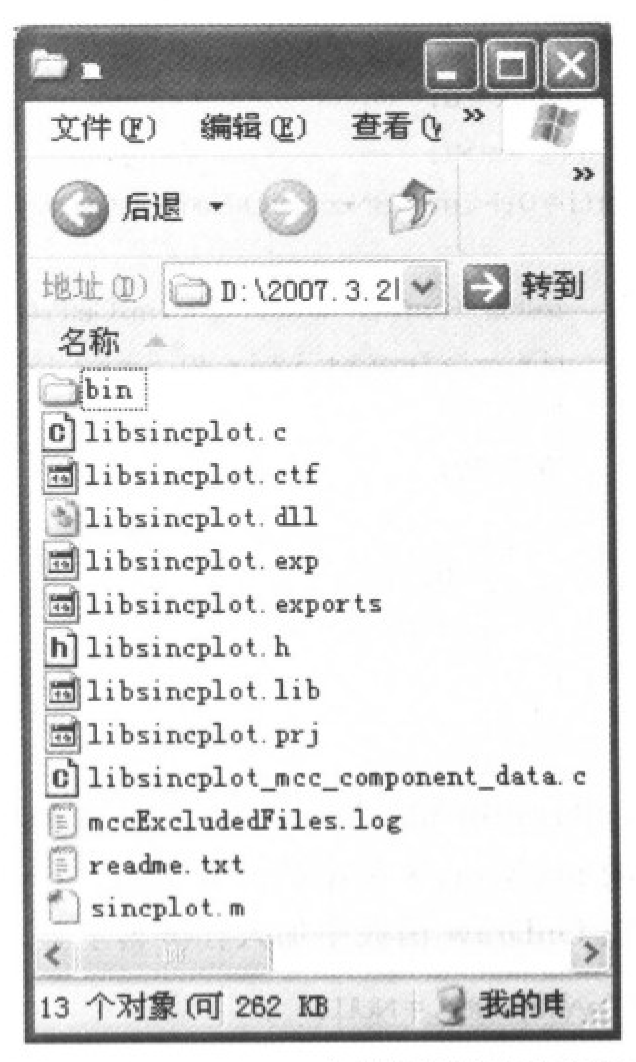


图 5-2　plotsinc. m 文件编译为动态链接库的过程中生成的文件列表

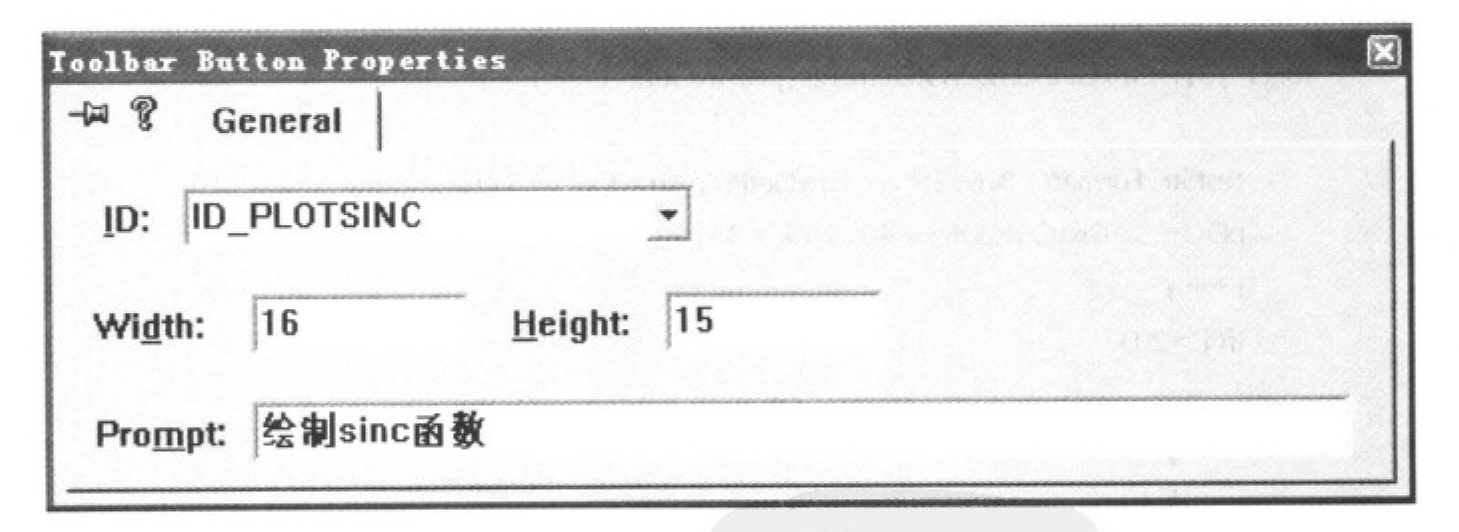


图 5-3　添加 ID_PLOTSINC 工具条按钮

在消息响应中加入调用 mlxSincplot 函数的代码如下所示。

```
if(pArrayIn==NULL)
{
    pArrayIn=mxCreateDoubleMatrix(1,1,mxREAL);
}
//构造一个[3 10]的随机的输入参数
*(mxGetPr(pArrayIn))=10*(0.3+0.7*rand()*1.0/RAND_MAX);
mlfSincplot(1,&pArrayOut,pArrayIn);
CDC* pDC=GetDC();
```

```
int i=0;
int j=0,k=0;
CString testStr;
for(i=0;i<mxGetNumberOfElements(pArrayOut);i++)
{
    testStr.Format(" %6.2f", *(mxGetPr(pArrayOut)+i));
    pDC->TextOut(5+j*40,5+k*15,testStr);
    j++;
    if(j>20)
    {
        j=0;
        k++;
    }
}
```

其中,pArrayIn 和 pArrayOut 为在 Testsincplot_dllView.h 中声明的 CTestsincplot_dllView 类中的 mxArray * 类型的公有变量。

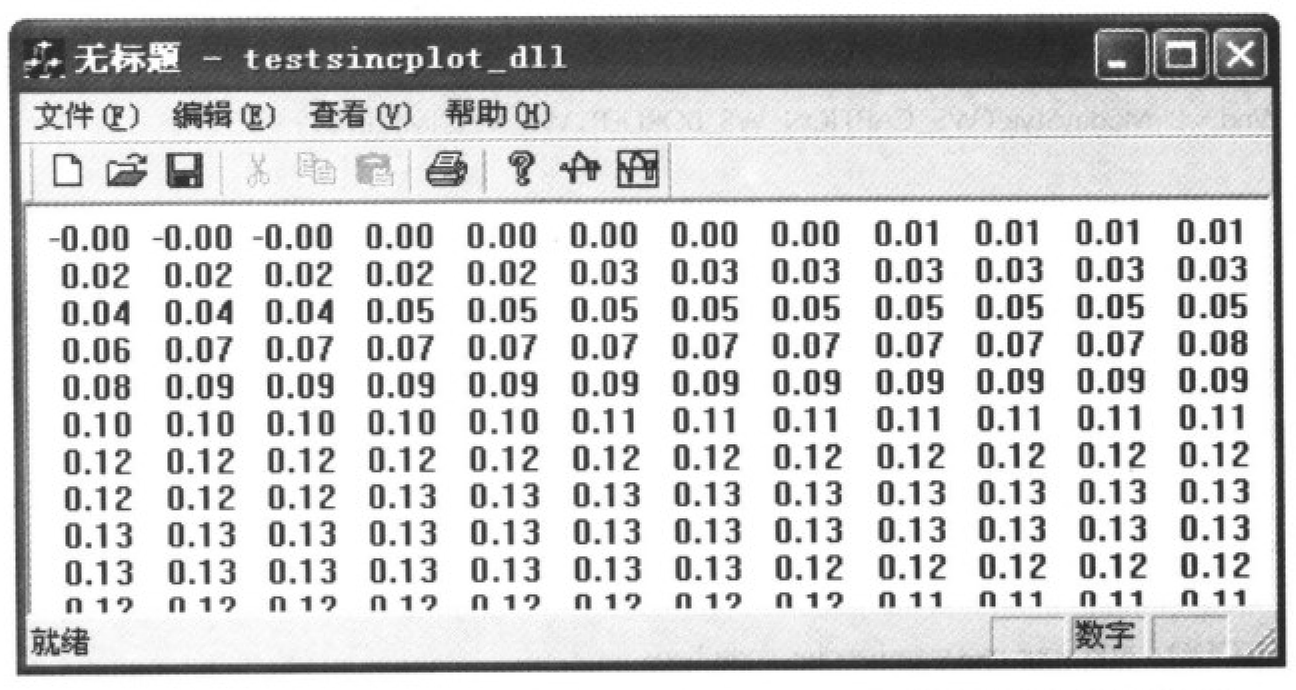
在 OnDraw 函数中加入在屏幕上显示 mlxSincplot 输出数据的代码如下所示。

```
if(pArrayOut! =NULL)
    {
        int i=0;
        int j=0,k=0;
        CString testStr;
        for(i=0;i<mxGetNumberOfElements(pArrayOut);i++)
        {
            testStr.Format(" %6.2f", *(mxGetPr(pArrayOut)+i));
            pDC->TextOut(5+j*40,5+k*15,testStr);
            j++;
            if(j>20)
            {
                j=0;
                k++;
            }
        }
    }
```

编译并运行其结果如图 5-4 所示。

此时还可以通过 FindWindow、ModifyStyle 和 MoveWindow 函数将生成的 Matlab 图形绘制窗口嵌入到 CTestsincplot_dllView 窗口中,这样在某些情况下可以使整个程序的运行风格更加协调。

为此,添加一个工具栏按钮 ID_PLOTINVIEW,并为 CTestsincplot_dllView 添加一个公有变量 HWND m_plotH,通过选择 View|ClassWizard 菜单项为 CTestsincplot_dllView 添加一个按钮 ID_PLOTINVIEW 的消息响应函数,并为此消息响应函数添加如下代码。



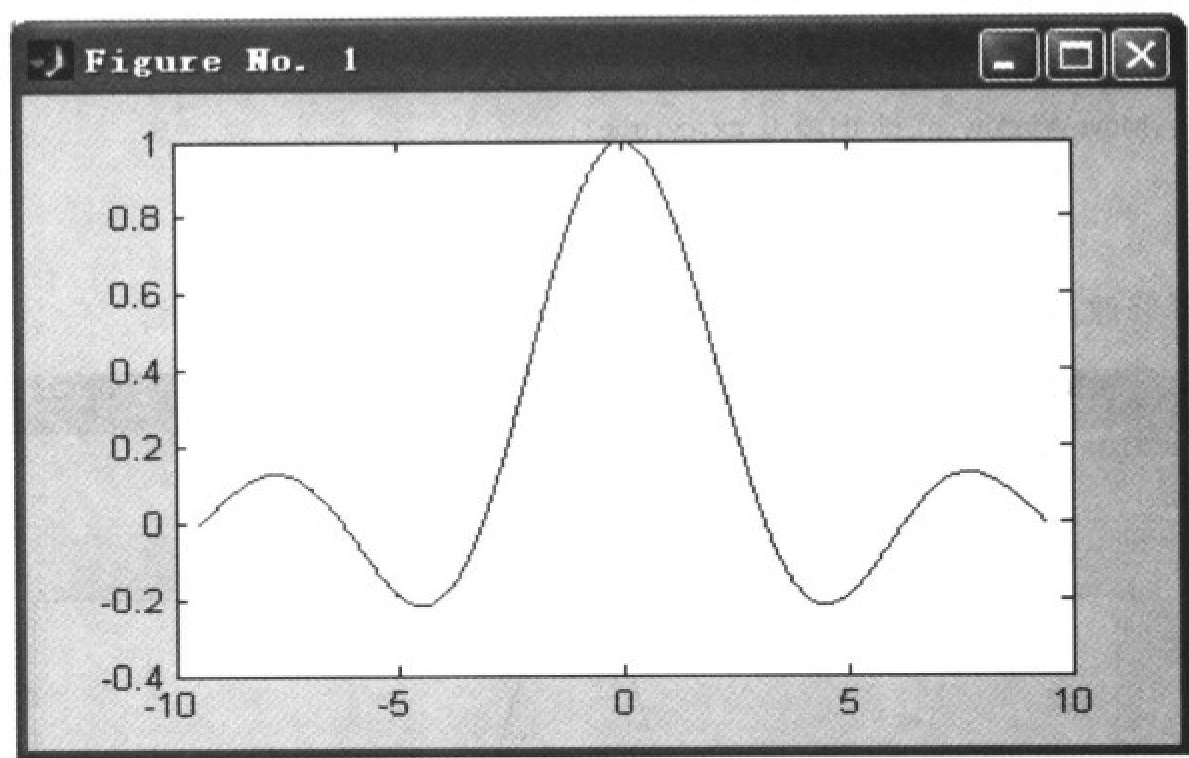


图 5-4 testsincplot_dll 工程执行结果

```
void CTestsincplot_dllView::OnPlotinview()
{
    // TODO: Add your command handler code here
    m_plotH=::FindWindow(NULL,"Figure No. 1");
    if(::IsWindow(m_plotH))
    {
        //将 CTestsincplot_dllView 设为 Matlab 窗口的父窗口
        ::SetParent(m_plotH,this->GetSafeHwnd());
        CWnd* pWnd=NULL;
        pWnd=FromHandle(m_plotH);
        CRect rect;
        GetClientRect(&rect);
        //改变 Matlab 绘图窗口的大小使其与父窗口客户区大小相同
        pWnd->MoveWindow(&rect,false);
```

```
        //修改 Matlab 绘图窗口的风格，去掉其边框和标题栏，并使其保持窗口的
        //最大化状态
    pWnd->ModifyStyle(WS_CAPTION|WS_BORDER,WS_MAXIMIZE,0);
    }
}
```

为了使 Matlab 窗口的大小能够随其父窗口客户区大小的变化而变化，重载 CTestsincplot_dllView WM_SIZE 消息的处理函数 OnSize。重载后的 OnSize 函数如下所示。

```
void CTestsincplot_dllView::OnSize(UINT nType,int cx,int cy)
{
    CView::OnSize(nType,cx,cy);

    // TODO: Add your message handler code here
    if(::IsWindow(m_plotH))
    {
        ::MoveWindow(m_plotH,0,0,cx,cy,true);
    }
}
```

重新编译修改后的 testsincplot_dll 工程，其运行结果如图 5-5 所示。

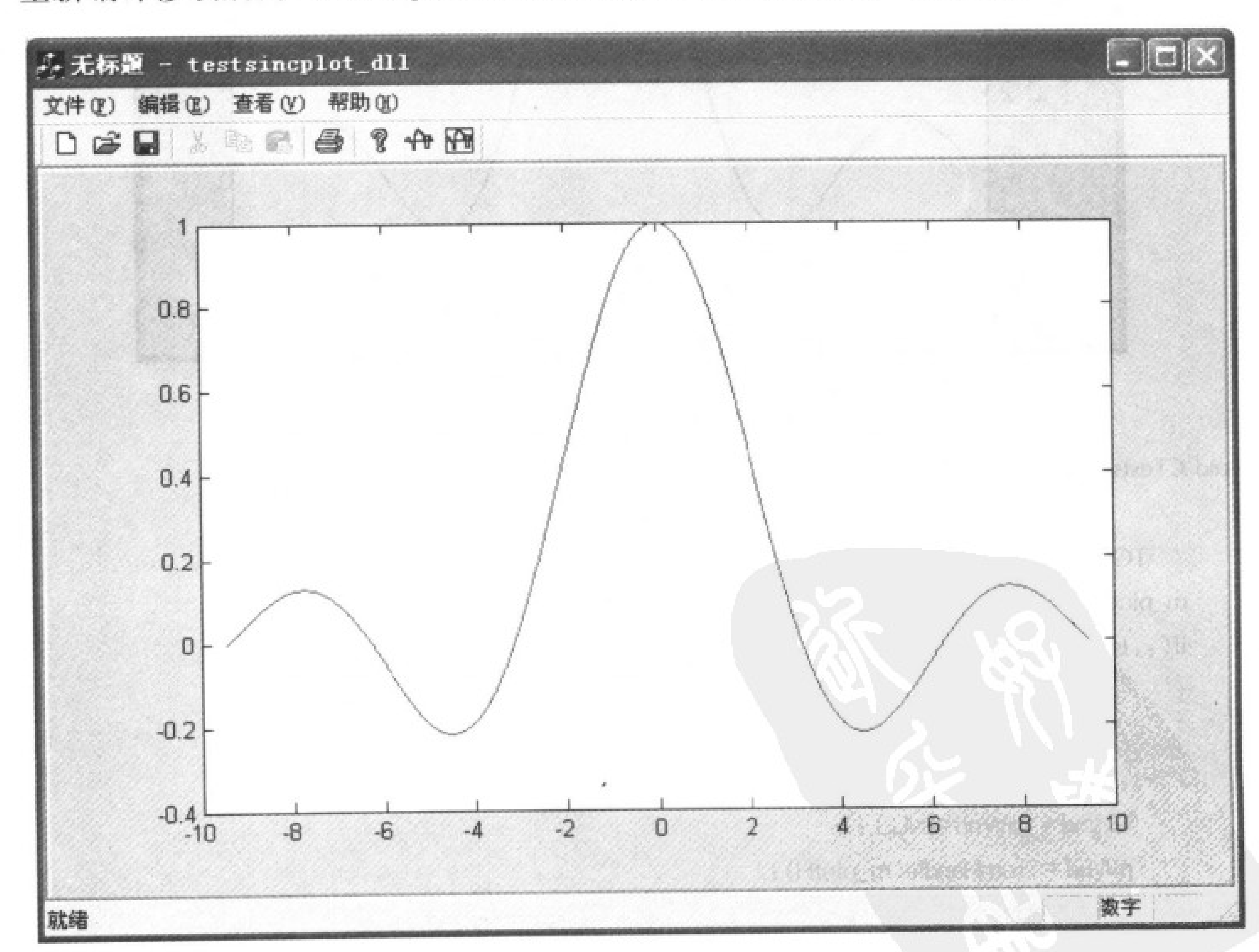


图 5-5　将 Matlab 窗口嵌入到 MFC View 窗口中

下面给出 testsincplot_dll 工程 CTestsincplot_dllView 的全部实现代码。

```
// testsincplot_dllView.h : interface of the CTestsincplot_dllView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_TESTSINCPLOT_DLLVIEW_H__53FF041D_20C9_45D2_AC6F_77E15108B51A__INCLUDED_)
#define AFX_TESTSINCPLOT_DLLVIEW_H__53FF041D_20C9_45D2_AC6F_77E15108B51A__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "libsincplot.h"

class CTestsincplot_dllView : public CView
{
protected: // create from serialization only
    CTestsincplot_dllView();
    DECLARE_DYNCREATE(CTestsincplot_dllView)

// Attributes
public:
    CTestsincplot_dllDoc* GetDocument();

// Operations
public:
    mxArray* pArrayIn;
    mxArray* pArrayOut;
    HWND m_plotH;
    int m_isInView;

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestsincplot_dllView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
    //}}AFX_VIRTUAL
```

```
// Implementation
public:
    virtual ~CTestsincplot_dllView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CTestsincplot_dllView)
    afx_msg void OnPlotsinc();
    afx_msg void OnPlotinview();
    afx_msg void OnSize(UINT nType,int cx,int cy);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in testsincplot_dllView.cpp
inline CTestsincplot_dllDoc* CTestsincplot_dllView::GetDocument()
   { return (CTestsincplot_dllDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif
// !defined(AFX_TESTSINCPLOT_DLLVIEW_H__53FF041D_20C9_45D2_AC6F_77E15108B51A__INCLUDED_)



// testsincplot_dllView.cpp : implementation of the CTestsincplot_dllView class
//

#include "stdafx.h"
#include "testsincplot_dll.h"

#include "testsincplot_dllDoc.h"
#include "testsincplot_dllView.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView

IMPLEMENT_DYNCREATE(CTestsincplot_dllView,CView)

BEGIN_MESSAGE_MAP(CTestsincplot_dllView,CView)
    //{{AFX_MSG_MAP(CTestsincplot_dllView)
    ON_COMMAND(ID_PLOTSINC,OnPlotsinc)
    ON_COMMAND(ID_PLOTINVIEW,OnPlotinview)
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT,CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT,CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW,CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView construction/destruction

CTestsincplot_dllView::CTestsincplot_dllView()
{
    // TODO: add construction code here
    libsincplotInitialize();
    this->pArrayIn = NULL;
    this->pArrayOut = NULL;
    m_isInView = 0;
}

CTestsincplot_dllView::~CTestsincplot_dllView()
{
    if(pArrayIn)
    {
        mxDestroyArray(pArrayIn);
    }
    if(pArrayOut)
```

```
        {
            mxDestroyArray(pArrayOut);
        }
        libsincplotTerminate();
}

BOOL CTestsincplot_dllView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView drawing

void CTestsincplot_dllView::OnDraw(CDC * pDC)
{
    CTestsincplot_dllDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    if(m_isInView == 1)
    {
        ;
    }
    else
    {
        if(pArrayOut! =NULL)
        {
            int i=0;
            int j=0,k=0;
            CString testStr;
            for(i=0;i<mxGetNumberOfElements(pArrayOut);i++)
            {
                testStr.Format("%6.2f", * (mxGetPr(pArrayOut)+i));
                pDC->TextOut(5+j*40,5+k*15,testStr);
                j++;
                if(j>20)
                {
                    j=0;
                    k++;
                }
```

```
                }
            }
        }
}

///////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView printing

BOOL CTestsincplot_dllView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CTestsincplot_dllView::OnBeginPrinting(CDC* /*pDC*/,CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CTestsincplot_dllView::OnEndPrinting(CDC* /*pDC*/,CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

///////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView diagnostics

#ifdef _DEBUG
void CTestsincplot_dllView::AssertValid() const
{
    CView::AssertValid();
}

void CTestsincplot_dllView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CTestsincplot_dllDoc* CTestsincplot_dllView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTestsincplot_dllDoc)));
    return (CTestsincplot_dllDoc*)m_pDocument;
}
```

```
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTestsincplot_dllView message handlers

void CTestsincplot_dllView::OnPlotsinc()
{
    // TODO: Add your command handler code here
    if(pArrayIn==NULL)
    {
        pArrayIn=mxCreateDoubleMatrix(1,1,mxREAL);
    }
    //构造一个[3 10]的随机的输入参数
    *(mxGetPr(pArrayIn))=10*(0.3+0.7*rand()*1.0/RAND_MAX);
    mlfSincplot(1,&pArrayOut,pArrayIn);
    CDC* pDC=GetDC();
    int i=0;
    int j=0,k=0;
    CString testStr;
    for(i=0;i<mxGetNumberOfElements(pArrayOut);i++)
    {
        testStr.Format("%6.2f",*(mxGetPr(pArrayOut)+i));
        pDC->TextOut(5+j*40,5+k*15,testStr);
        j++;
        if(j>20)
        {
            j=0;
            k++;
        }
    }
}

void CTestsincplot_dllView::OnPlotinview()
{
    // TODO: Add your command handler code here
    m_plotH=::FindWindow(NULL,"Figure No. 1");
    if(::IsWindow(m_plotH))
    {
        //将 CTestsincplot_dllView 设为 Matlab 窗口的父窗口
        ::SetParent(m_plotH,this->GetSafeHwnd());
        CWnd* pWnd=NULL;
        pWnd=FromHandle(m_plotH);
        CRect rect;
```

```
        GetClientRect(&rect);
        pWnd->MoveWindow(&rect,false);
        pWnd->ModifyStyle(WS_CAPTION|WS_BORDER,WS_MAXIMIZE,0);
        m_isInView = 1;
        Invalidate(TRUE);
    }
}

void CTestsincplot_dllView::OnSize(UINT nType,int cx,int cy)
{
    CView::OnSize(nType,cx,cy);

    // TODO: Add your message handler code here
    if(::IsWindow(m_plotH))
    {
        ::MoveWindow(m_plotH,0,0,cx,cy,true);
    }
}
```



第 6 章　Matlab Dotnet Builder 与 Visual C++

6.1　COM 基础知识

6.1.1　COM 组件概述

COM(Componet Object Model)是以组件为发布单元的对象模型。由于 COM 是建立在二进制级别上的规范，所以组件对象之间的交互规范不依赖于任何特定的开发语言。COM 用于不同语言的协作开发是非常方便的。COM 开发架构是以组件为基础的，可以把组件看做是用于"搭建"软件的积木块，采用这种开发模式除了跨语言的特性以外，还可以带来很多好处，例如采用组件替换可以使得软件系统的升级换代更加简单，可以在多个不同的软件开发应用中重复利用同一个组件等。

如图 6-1 (a)所示，开发人员可以采用不同的开发语言开发组件 A、B、C、D、E 等，通过组件开发模式，可以像搭积木一样将它们组合成一个具有完整功能的应用软件。当开发人员发现组成应用软件的某些组件模块需要更新时，又可以像替换某块积木块一样将需要更新的组件替换下来，如图 6-1 (b)所示。不仅如此，一旦组件开发完成以后，通过将不同组件的重新调整和整合，可以重复利用组件开发不同的应用软件，如图 6-1 (c)所示。

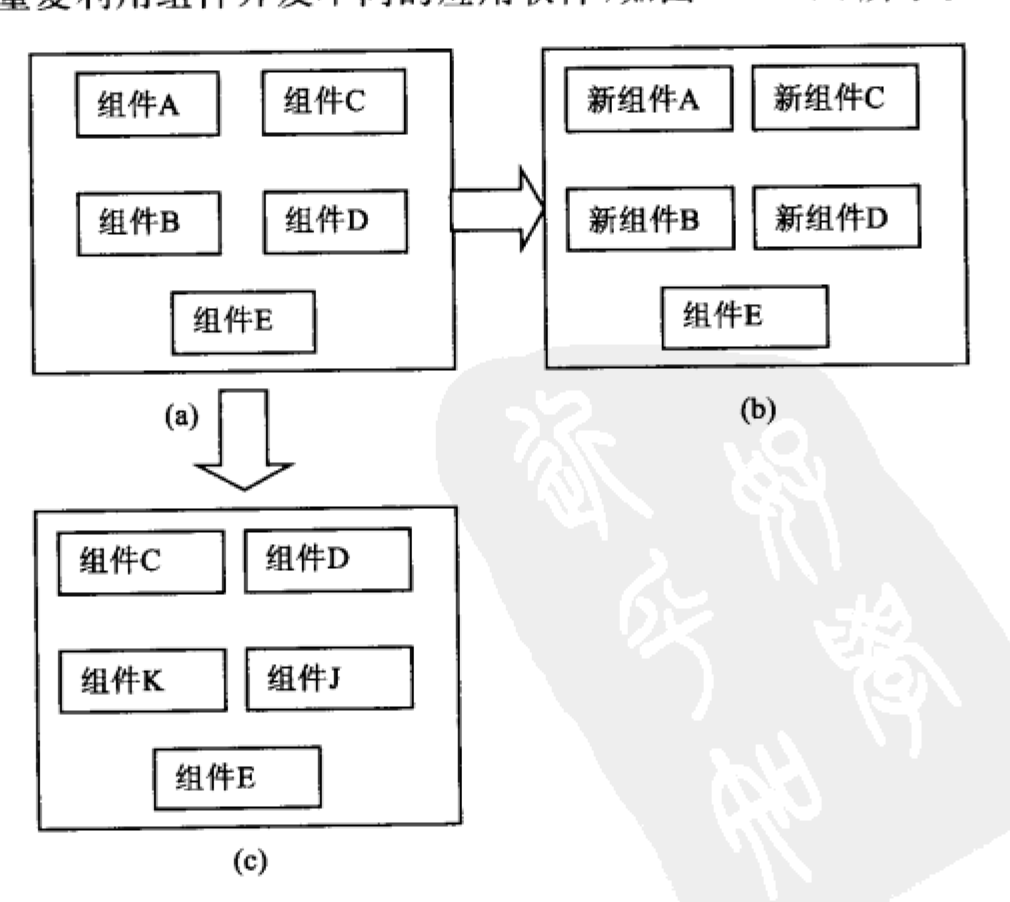


图 6-1　利用组件进行开发的示意图

6.1.2　COM 组件开发的基础知识

1. 对象和接口

COM 是面向对象的开发模型，COM 对象的概念与 C++对象的概念相似。面向对象的开发模式是将数据和功能按照某种意义组合到一起形成类，对象就是类的实例。面向对象的开发模式更加贴近人们对事物的认识。

接口是一组函数的集合。接口是 COM 对象与外界交互的唯一方式，因为 COM 对象是对现实世界某种事物的抽象，所以其接口的所有函数在某种程度上有其相关性。

如图 6-2 所示，COM 组件由一个或多个 COM 对象构成，COM 对象通过接口与使用 COM 的客户进行交互。使用 COM 组件的程序被称为 COM 客户，COM 对象对于 COM 客户来说是不可见的，因而 COM 客户只能通过接口来访问 COM 对象。

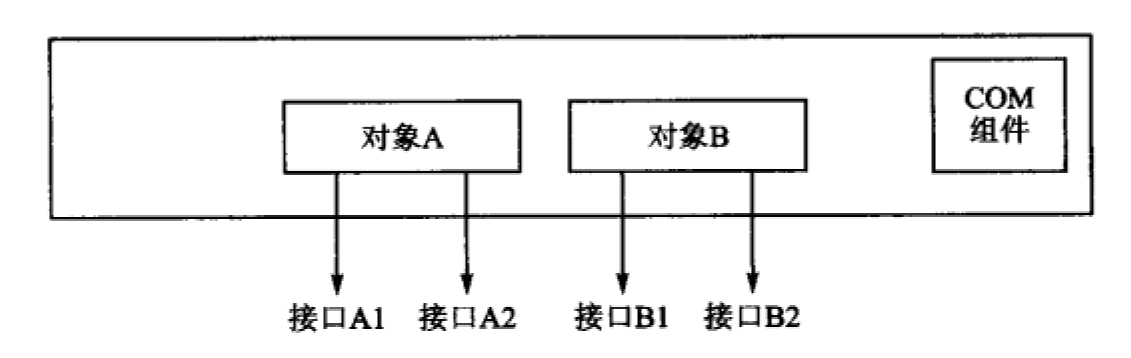


图 6-2　COM 对象与接口

2. COM 在 Windows 平台下的实现方式

(1) COM 的存在形式

COM 最早应用在 OLE 中，由于 COM 开发模型特有的优点，现在 COM 已经渗透到了 Windows 的各个角落，为人熟知的 DriectX 多媒体开发包就建立在 COM 组件技术之上。COM 在 Windows 平台上以动态链接库的形式存在。采用动态链接库可以由 COM 组件客户来决定什么时候加载所需要的 COM 组件。

(2) COM 组件的标志方式

但是，由于 COM 组件的类别多种多样，开发厂商也有很多，如何来区别这众多的 COM 组件呢？如果采用命名的方式，免不了有很多重复的情况发生，因而 COM 规范采用一个 128 位长度的常量来标志 COM 组件。由于这个常量的位数过长，开发语言的编译器一般都不支持这么长的数据类型，因而采用一个结构体来表示这 128 位数：

```
typedef struct _GUID {
    DWORD Data1;
    WORD    Data2;
    WORD    Data3;
    BYTE   Data4[8];
} GUID;
```

同样，为了唯一地标志组件的接口，COM 规范也采用这种 128 位 GUID 来作为接口的标志。为了区分组件和接口的标志，一般将 COM 组件的 GUID 标志称为 CLSID，将接口的标志称为 IID。打开 VC++ 6.0 的 include 目录中的 mtypes.h，可以找到下面的定义：

```
typedef GUID CLSID;
typedef GUID IID;
```

接下来的问题是各地开发者都在使用 GUID,如何生成唯一不重复的 GUID 呢？这个也不用担心,Microsoft VC++提供的 UUIDGEN. exe 和 GUIDGEN. exe 有助于完成这个工作。而且,采用这种方法可以保证 3400 年以前生成的 GUID 不会有重复的可能。在 VC++中,接口的 IID 和 COM 组件的 CLSID 一般采用如下的方法定义：

```
const IID IID_IMyCom =
        {0x0F29F908,0x00BF,0x4A0C,{0xB1,0xED,0x44,0xE3,0xD0,0x70,0x24,0xA2}};
const CLSID CLSID_MyCom =
      {0xC7C0BFA5,0xBFC9,0x41A4,{0x8E,0xC7,0x60,0x5D,0xB8,0x3E,0xEB,0xD0}};
```

GUID 记忆起来很困难,所以有些计算机语言采用 ProgID 来标志组件,ProgID 是指定给某个组件的友好的名字,ProgID 应该与 CLSID 一一对应,但是由于是一种人为命名的方法,因而 ProgID 的唯一性有可能得不到保证。

(3) COM 组件与注册表

客户需要创建 COM 组件时,需要知道 COM 组件相对应的动态链接库的存在位置。在 Windows 操作系统中,注册表是组件客户和组件都能够访问的共享信息。因此,COM 组件通过注册将其对象和接口信息存放到注册表中,这样客户创建组件时就可以通过访问注册表来获取其需要的信息。在 Windows 中单击"开始""|运行"系统菜单,输入 regedit 命令打开注册表编辑器,搜索 MWComUtil,可以找到 MWComUtil. MWUtil7. 6,这就是 MWComUtil 组件的 ProgID,其 CLSID 如图 6－3 所示。

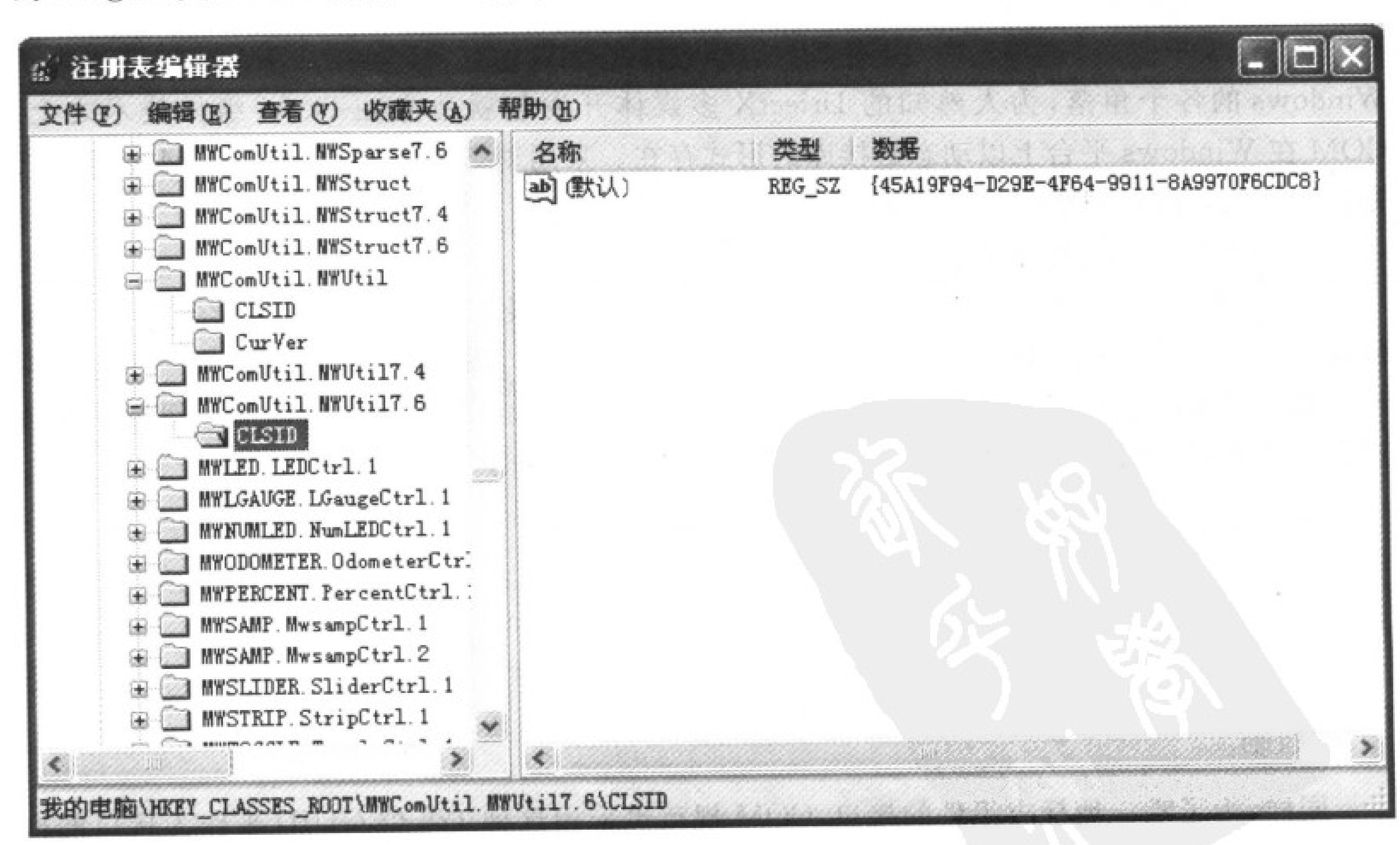


图 6－3　注册表中的 ProgID 和 CLSID

3. VC++中调用 COM 组件

(1) HRESULT 类型

COM 组件的接口函数用 HRESULT 来向其客户报告执行情况，HRESULT 不同于布尔值，一般不能采用直接比较的方法来判断某个函数是否执行成功，如下面的做法就可能带来问题。

```
...
HRESULT hr;
hr = IMyInterface->MyFun(…);
if(hr == E_FAIL)//不要采用这种做法
{
    return;
}
```

在 COM 中，采用 SUCCEEDED 和 FAILED 宏来判断函数是否执行成功，如：

```
...
HRESULT hr;
hr = IMyInterface->MyFun(…);
if(FAILED(hr))//正确的做法
{
    return;
}
```

HRESULT 与布尔值不同的地方在于，查函数执行失败以后，采用 HRESULT 可以返回多种失败的原因。在程序调试时，往往需要知道 COM 失败的具体原因，可以通过下面的方式显示 COM 的错误信息。

```
void ErrorMessage(LPCTSTR str,HRESULT hr)
{
    void * pMsgBuf;
    ::FormatMessage(
            FORMAT_MESSAGE | FORMAT_MESSAGE_FROM_SYSTEM,
            NULL,
            hr,
            MAKEFAILED(LANG_NEUTRAL,SUMLANG_DEFAULT),
            (LPTSTR) &pMsgBuf,
            0,
            NULL);
    cout << str <<"\r \n";
    cout<<"Error(" << hex <<hr<<"):"
        <<(LPTSTR)pMsgBuf<<endl;
    LocalFree(pMsgBuf);
}
```

(2) 字符串

因为 COM 要实现独立于语言的高度通用性，所以 COM 采用 UNICODE 字符串。下面

是几种常用的 UNICODE 字符类型的声明：

```
typedef WCHAR OLECHAR;
typedef wchar_t WCHAR;
```

要实现从 ANSI 到 Unicode 的转换，可以采用 MultiByteToWideChar 函数和 A2W 宏，反之则采用 WideCharToMultiByte 函数和 W2A 宏来实现。

(3) CLSID 和 ProgID 的相互转换

CLSIDFromProgID 用于通过 COM 组件的 ProgID 获取其 CLSID，其函数声明如下所示。

```
HRESULT CLSIDFromProgID(
        LPCOLESTR lpszProgID,    //Pointer to the ProgID
        LPCLSID pclsid           //Pointer to the CLSID
    );
```

ProgIDFromCLSID 用于通过 COM 组件的 CLSID 获取其 ProgID，其函数声明如下所示：

```
HRESULT ProgIDFromCLSID(
                REFCLSID clsid,    //CLSID for which the ProgID is requested
                LPOLESTR * lplpszProgID
                   //Address of output variable that receives a
                   // pointer to the requested ProgID string
        );
```

(4) AddRef()、Release()、QueryInterface()

所有的 COM 接口都是从接口 IUnknown 派生来的，IUnknown 接口的定义如下。

```
interface IUnknown
{
    virtual HRESULT __stdcall QueryInterface(const IID & iid,void ** ppv) = 0;
    virtual ULONG __stdcall AddRef() = 0;
    virtual ULONG __stdcall Release() = 0;
}
```

所以，所有的 COM 接口都至少要实现 AddRef、Release 和 QueryInterface 三个函数。

AddRef 和 Release 用于控制 COM 对象的生命周期，即 COM 从创建到销毁的时间。由于 COM 组件是在需要时通过动态链接库动态地加载的，COM 对象客户在创建 COM 组件对象时并不能保证只有一个客户在使用 COM 组件对象，所以当 COM 对象客户在对 COM 组件对象访问完毕后，也不应该马上销毁 COM 组件，因为此时很可能还有其他客户在使用此 COM 对象。因此 COM 就采用引用技术机制来实现其生命周期的自我控制。当增加一个客户时，COM 对象的引用计数就增加 1，当减少一客户时，COM 对象的引用计数就减少 1；只有当 COM 对象的引用计数减为 0 时，COM 才会自己销毁自己。客户从组件获得一个接口时，调用 AddRef，用于增加一个引用计数；当客户使用完某个接口以后，调用 Release，用于减少一个引用计数。

COM 组件中的接口数目可能有很多，如何才能获得客户需要的接口呢？由于 COM 组件

与客户的所有交互都是通过接口来实现的，因而任何接口的获得只能通过接口来实现。QueryInterface 函数就是用来查询组件是否支持某个特定的接口的。当然，接口查询的依据就是接口的标志 IID。根据 COM 规范，客户可以通过 QueryInterface 函数获得任何它想要的 COM 组件提供的接口。查询接口的示例程序如下。

```
//定义接口
IMyInterface * pInterface = NULL;
//创建接口和其他操作省略
...
//定义要查询的接口
IMyQueryInterface * pQueryInterface = NULL;
HRESULT hr;
hr = pInterface -> QueryInterface(IID_MyQueryInterface, (void ** )&pQueryInterface);

if(SUCCEEDED(hr))
{
    //使用这个接口
    ...
}
...
```

(5) 在 Visual C 中调用 COM 组件的步骤

Visual C 中调用 COM 组件的步骤如下所述。

①初始化 COM 库。

②得到 COM 对象的 CLSID。

③创建一个 COM 对象的实例。

④使用 COM 对象。

⑤退出 COM 库。

```
//初始化 COM 库
HRESULT CoInitialize(
            LPVOID pvReserved    //Reserved; must be NULL
            );

//根据返回的 HRESULT 值来判断 COM 库的初始化是否成功
...
HRESULT hr;
hr = CoInitialize(NULL);
if (FAILED(hr))
{
    return false;
}
...
```

```
//得到 COM 对象的 CLSID
CLSID CLSID_MyInterface;
HRESULT hr;
hr = CLSIDFromProgID(L"Myinterface.Myinterface1.0",&CLSID_MyInterface);
if (FAILED(hr))
{
        MessageBox("CLSIDFromProgID 调用失败");
        return false;
}

//创建一个 COM 对象的实例
IMyInterface * pIMyInterface;
//IID_IMyInterface 接口 IMyInterface 的 IID
hr = CoCreateInstance(CLSID_MyInterface,NULL,CLSCTX_ALL,
            IID_IMyInterface,(void ** )&pIMyInterface);
if (FAILED(hr))
{
      MessageBox("创建 IMyInterface 接口失败!");
      return false;
}


//使用创建的 COM 对象
...
//退出 COM 库
CoUninitialize()
```

6.2 DotnetBuilder 基础知识

6.2.1 配置 Matlab C/C++编译器

如果在使用 Matlab DotnetBuilder 以前还没有配置 Matlab C/C++编译器的话,需要首先运行:

```
>> mbuild  -setup
```

然后再配置 Matlab C/C++编译器。

6.2.2 使用 Matlab DotnetBuilder

在 Matlab 命令行通过 dotnettool 启动 Matlab DotnetBuilder 图形用户界面(通过 deploytool 也可以实现 Dotnet Builder 的功能),如图 6-4 所示。

采用 Matlab DotnetBuilder 的第一个步骤就是创建一个新的工程,在 Matlab DotnetBuilder 中选

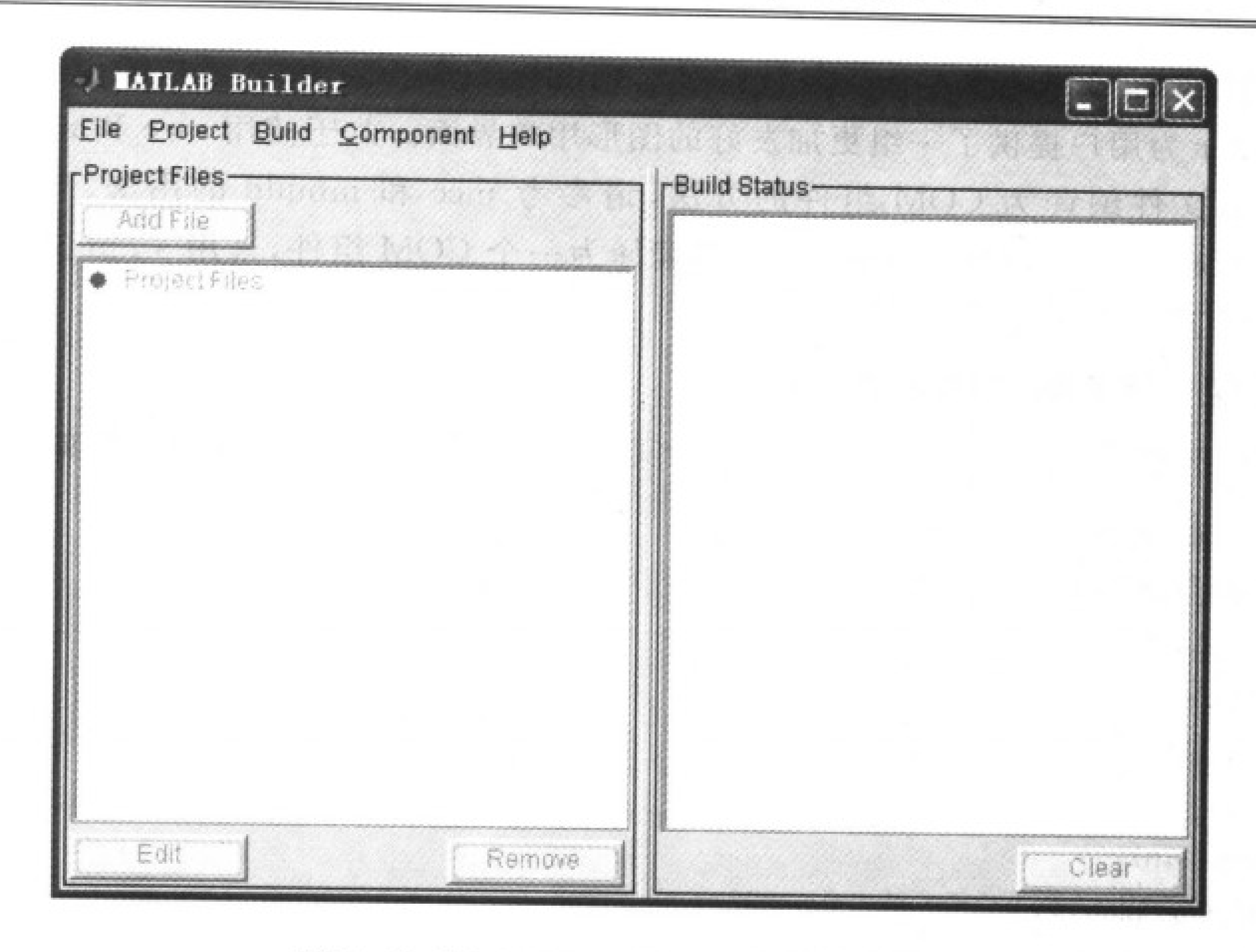


图 6－4　Matlab DotnetBuilder 图形用户界面

择 File｜New Project 菜单项，然后在弹出的 New Project Settings 对话框中填入相应的选项，如图 6－5 所示。

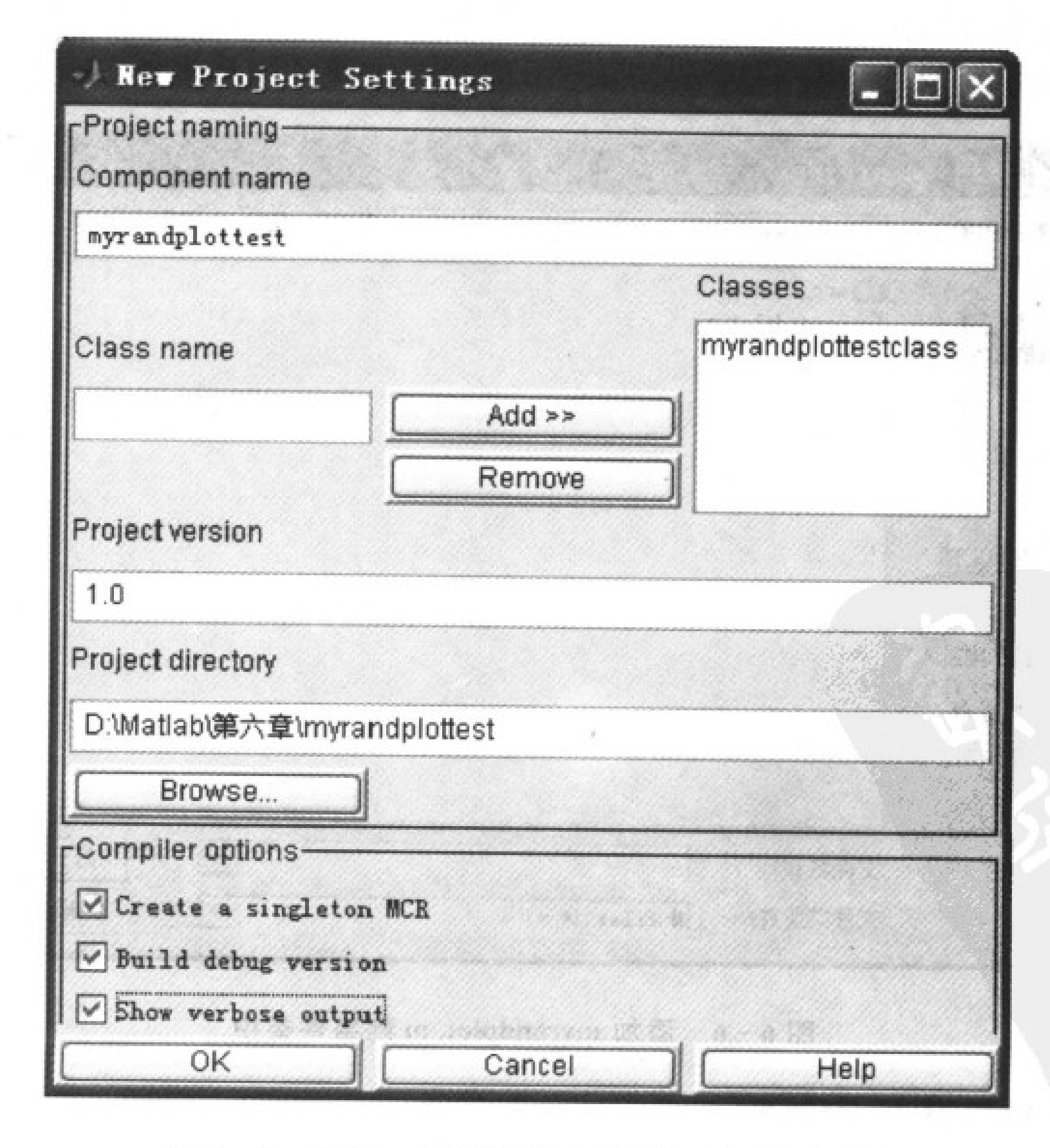


图 6－5　建立一个新的 Matlab DotnetBuilder 工程

其实采用 mcc 和 mbuild 命令完全可以实现 DotnetBuilder 的全部功能,只不过 Matlab DotnetBuilder 为用户提供了一组更加友好的图形用户界面。关于使用 mcc 和 mbuild 实现将 Matlab *.m 文件编译为 COM 组件的方法,请参考 mcc 和 mbuild 的用法。采用 Dotnet-Builder Builder 可以实现将多个 *.m 文件编译为一个 COM 组件,采用 *.m 文件可以实现 COM 组件的方法、属性和事件。

1. 实现 COM 组件的方法

COM 组件的方法可以用 Matlab 函数来实现,首先建立一个 Matlab 函数,如:

```
function [out] = myrandplot(n)
% function [out] = myrandplot(n)
global randplotcolor;
if length(n) > 1
    n = n(1);
end
out = rand(1,n);
if isempty(randplotcolor)
randplotcolor = [0 0 1];
end
plot(out,'color',randplotcolor);
```

然后将其保存为 myrandplot.m 文件,再选择 Project | Add File 菜单项或者直接单击工具栏上的 Add File 工具按钮,将 myrandplot.m 添加到新建的组件类中,如图 6-6 所示。

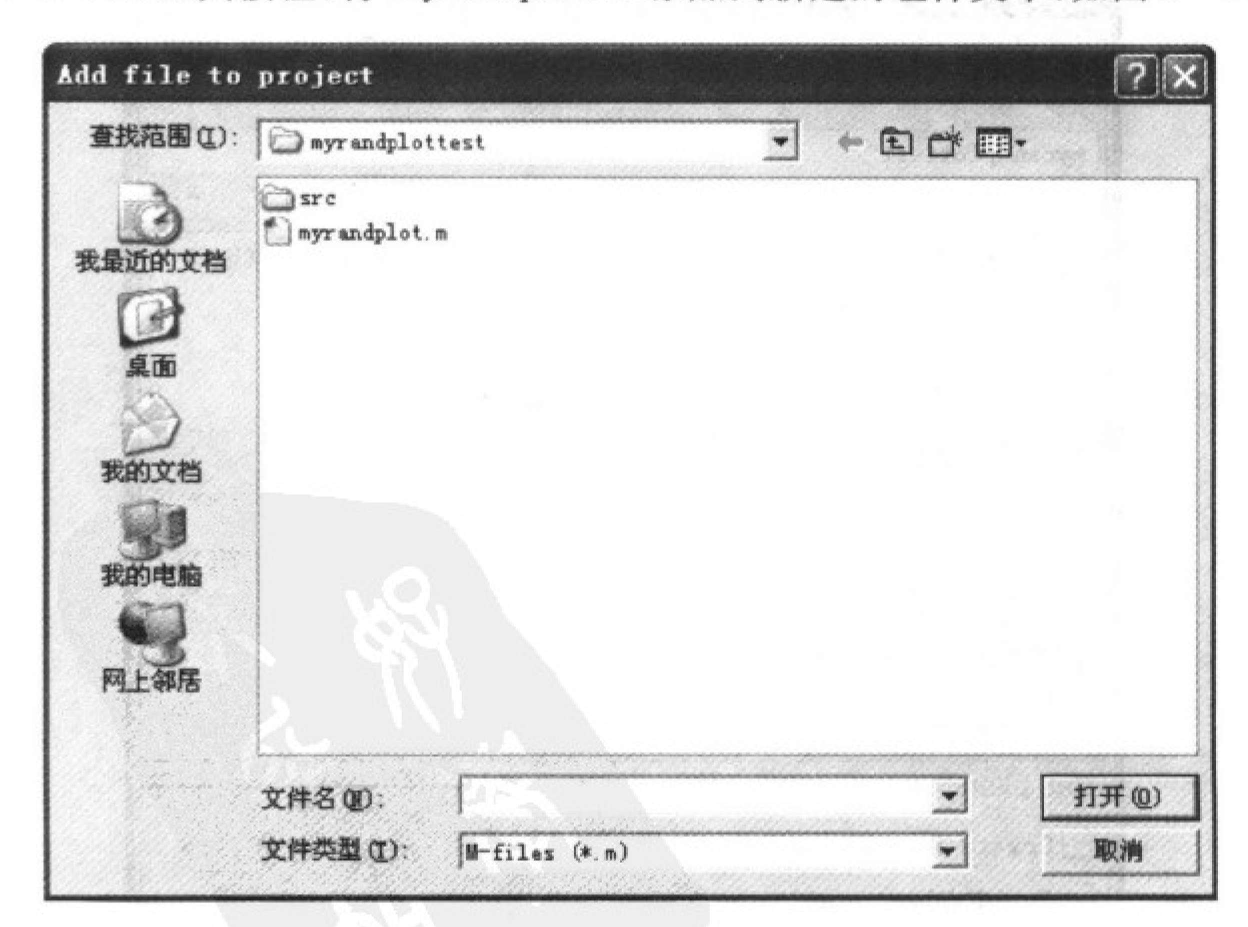


图 6-6 添加 myrandplot.m 到组件类中

2. 实现 COM 组件的属性

COM 组件的属性可以通过声明 global 变量实现,在 myrandplot 中声明了一个 global 变

量 randplotcolor。另外，还需要添加 SetRandPlotColor 和 GetRandPlotColor 两个函数实现对 COM 组件属性的访问。

```
function [out] = getplotcolor()
% function [] = setplotcolor(color)
global randplotcolor;
out = randplotcolor;

function [] = setplotcolor(color)
% function [] = setplotcolor(color)
global randplotcolor;
randplotcolor = color;
```

然后选择通过 Project | Add File 菜单项或者直接单击工具栏上的 Add File 工具按钮，将 getplotcolor. m 和 setplotcolor. m 添加到新建的组件类中。

3. 编译 COM 组件

通过选择 Build | COM Object 菜单项编译当前工程的 COM 组件，如图 6-7 所示。

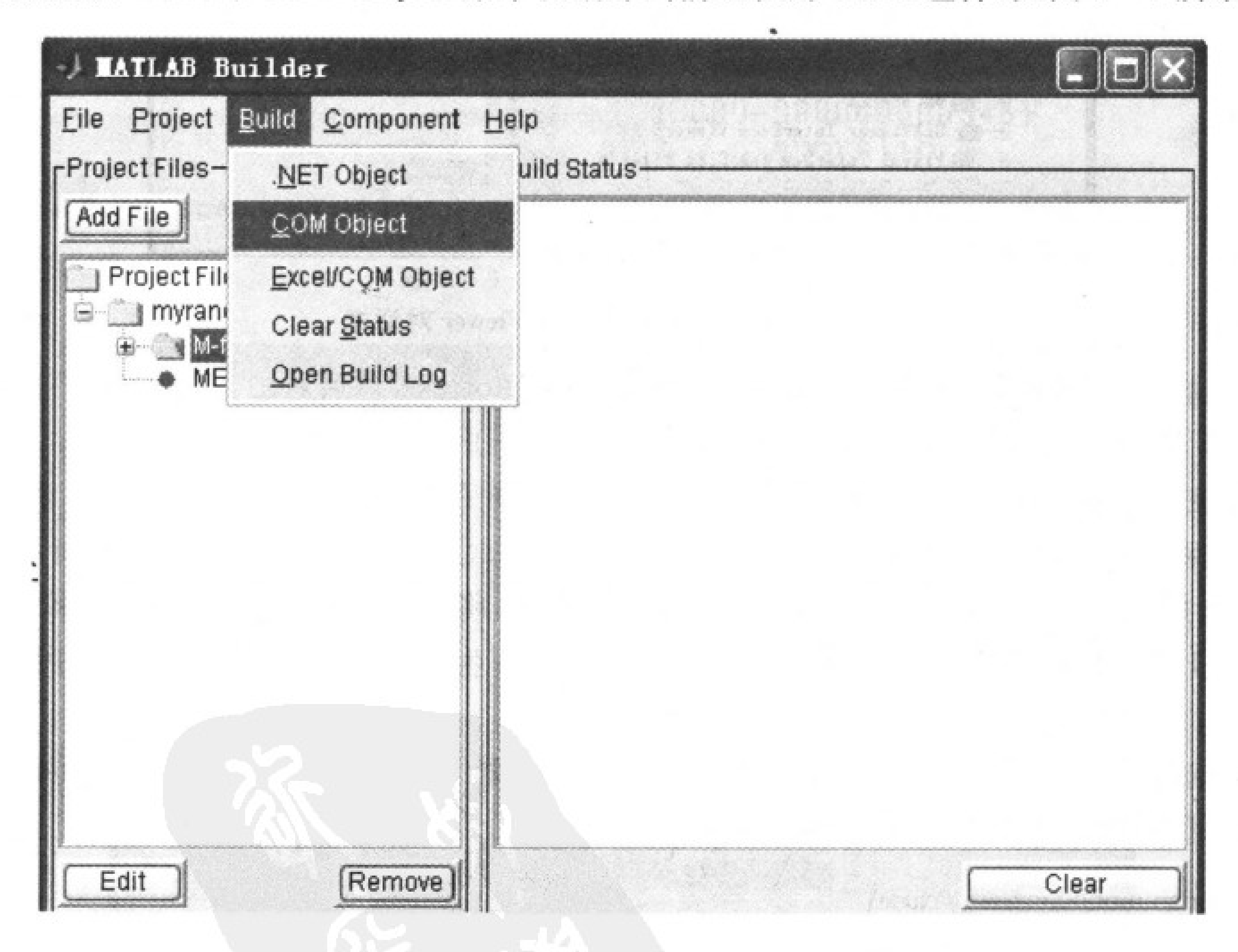


图 6-7　DotnetBuilder 编译 COM 组件

6.3　Visual C 调用 DotnetBuilder 生成的组件

Visual C 调用 Dotnet Builder 生成组件的过程如下所述。

① 建立一个 VC++ 6.0 MFC 对话框工程 myrandplotestvc。

② 在 VC++6.0 中选择 Tools | OLE/COM Object Viewer 菜单项，出现如图 6-8 所示的对话框。

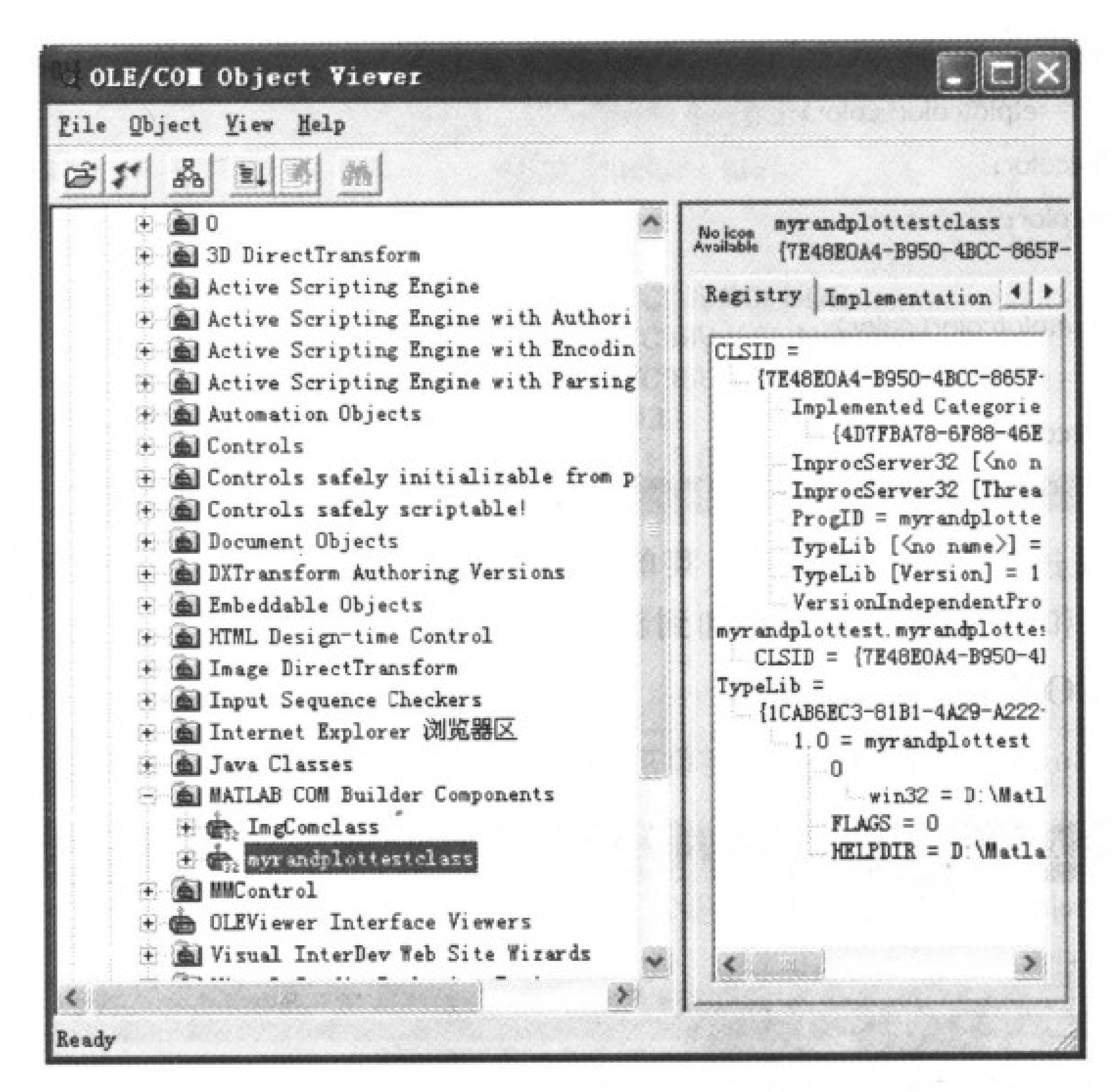


图 6-8　OLE/COM Object Viewer 对话框

③ 选中新建 Matlab Dotnet Builder 组件 myrandplottest，然后右击，在弹出的右键快捷菜单中选择 View Type Information 菜单项，出现如图 6-9 所示的对话框。

④ 在 ITypeLib Viewer 对话框中选择 File | Save As 菜单项，然后选择 *.h 和 *.c 文件，分别保存为 myrandplottest_1_0.h 和 myrandplottest_1_0.c 文件，如图 6-10 所示。

⑤选择 Project | Add To Project | Files 菜单项，将 myrandplottest_1_0.h 和 myrandplottest_1_0.c 文件添加到工程中。同时，将 src 目录下生成的 mwcomtypes.h 文件添加到工程中。

⑥ 选择 Tools | Options | Directories 菜单项，设置 VC++ 使用 Matlab Dotnet Builder 生成的 COM 组件需要的头文件路径，如图 6-11 所示。在 include files 中加入：

```
<Matlab root>\extern\include\
<Matlab root>\extern\include\win32
```

在 lib files 中加入下面两个目录：

```
<Matlab root>\extern\lib\win32\microsoft
<Matlab root>\ extern\lib\win32\microsoft \ Debug
```

⑦ 设置工程 Myrandplottestvc：选择 Project | Settings | C++ | Precompiled Headers 菜单项，在弹出的 Project Settings 对话框的 C/C++选项卡中选中 Automatic Use of pre-

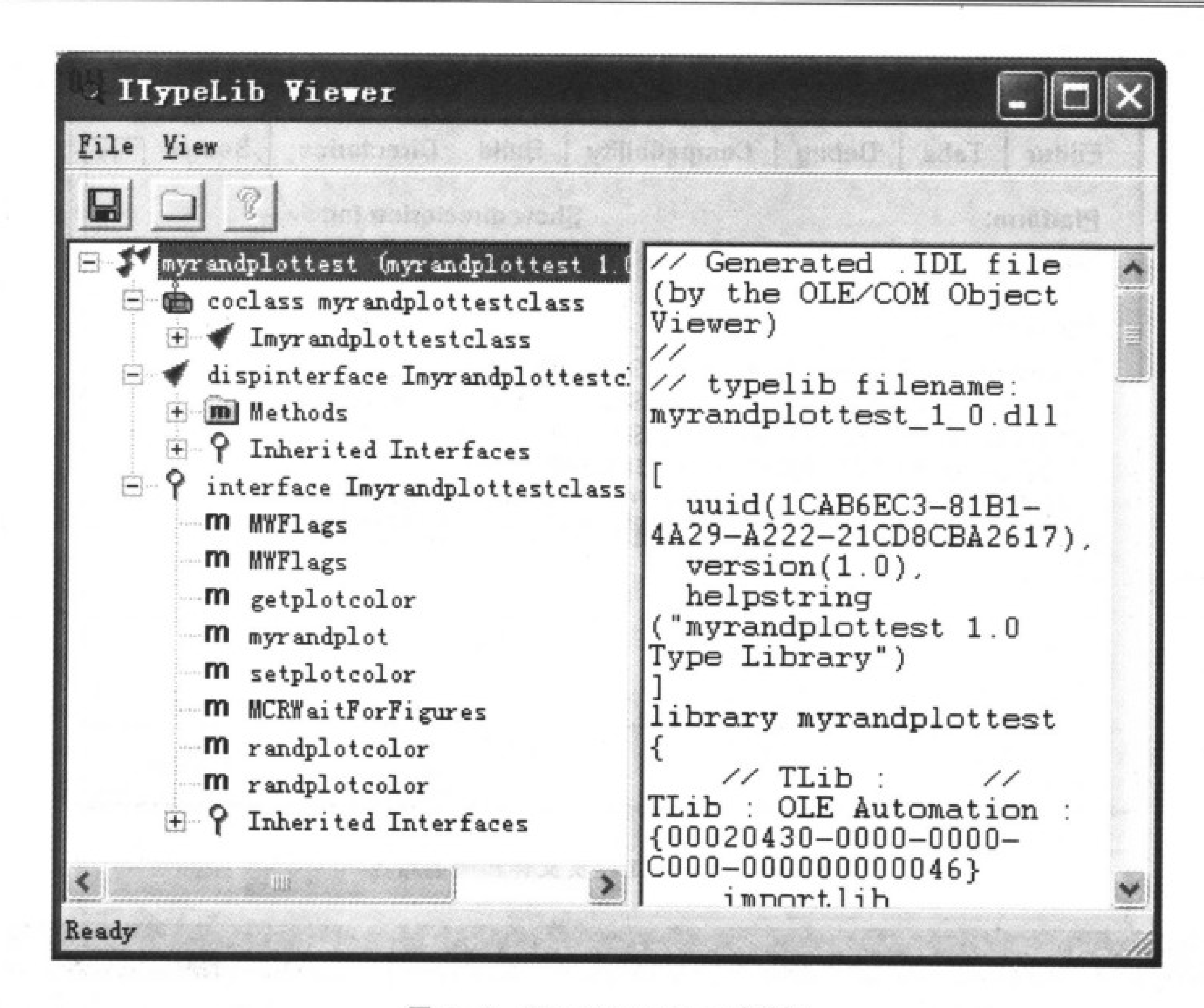


图 6-9　ITypeLib Viewer 对话框

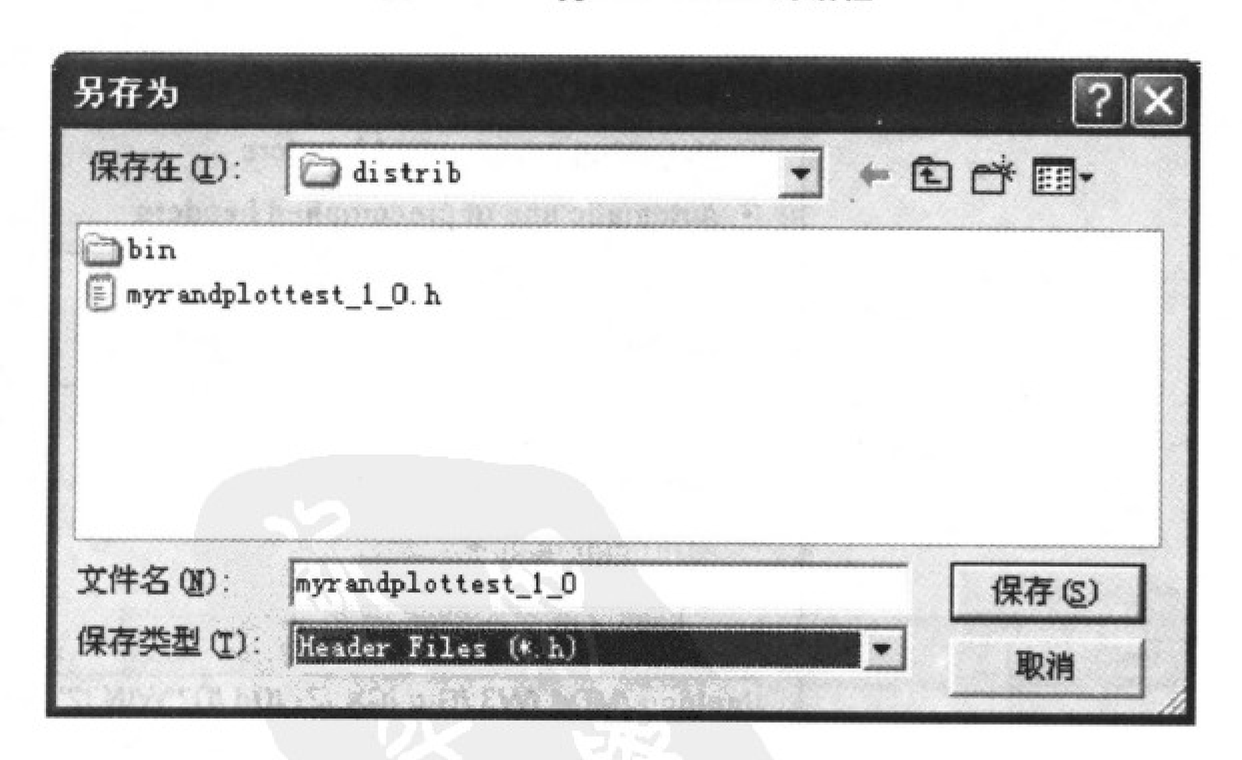


图 6-10　ITypeLib Viewer 中保存程序对话框

compiled headers 单选项，并将 header 设为 stdafx.h，如图 6-12 所示。

⑧ 在 CMyrandplottestvcDlg 中加入公共变量 Imyrandplottestclass * m_pTest。

⑨ 在 CMyrandplottestvcDlg 的 OnInitDialog() 函数中加入调用 COM 组件的初始化代码。

⑩ 在编辑 CMyrandplottestvcDlg 的对话框资源，添加 IDTEST(测试)和 IDQUIT(退出)

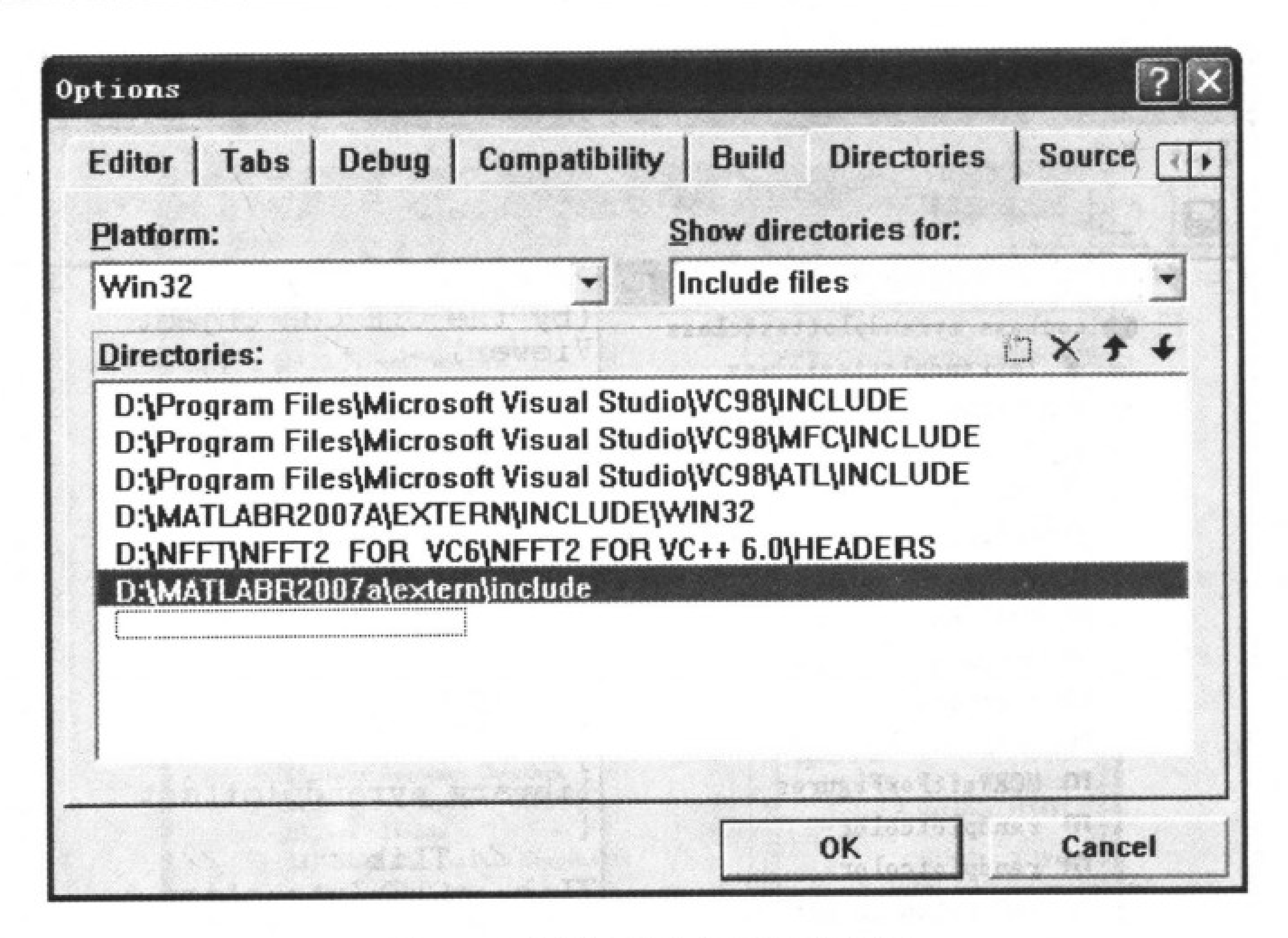


图 6-11　设置工程头文件和库的路径

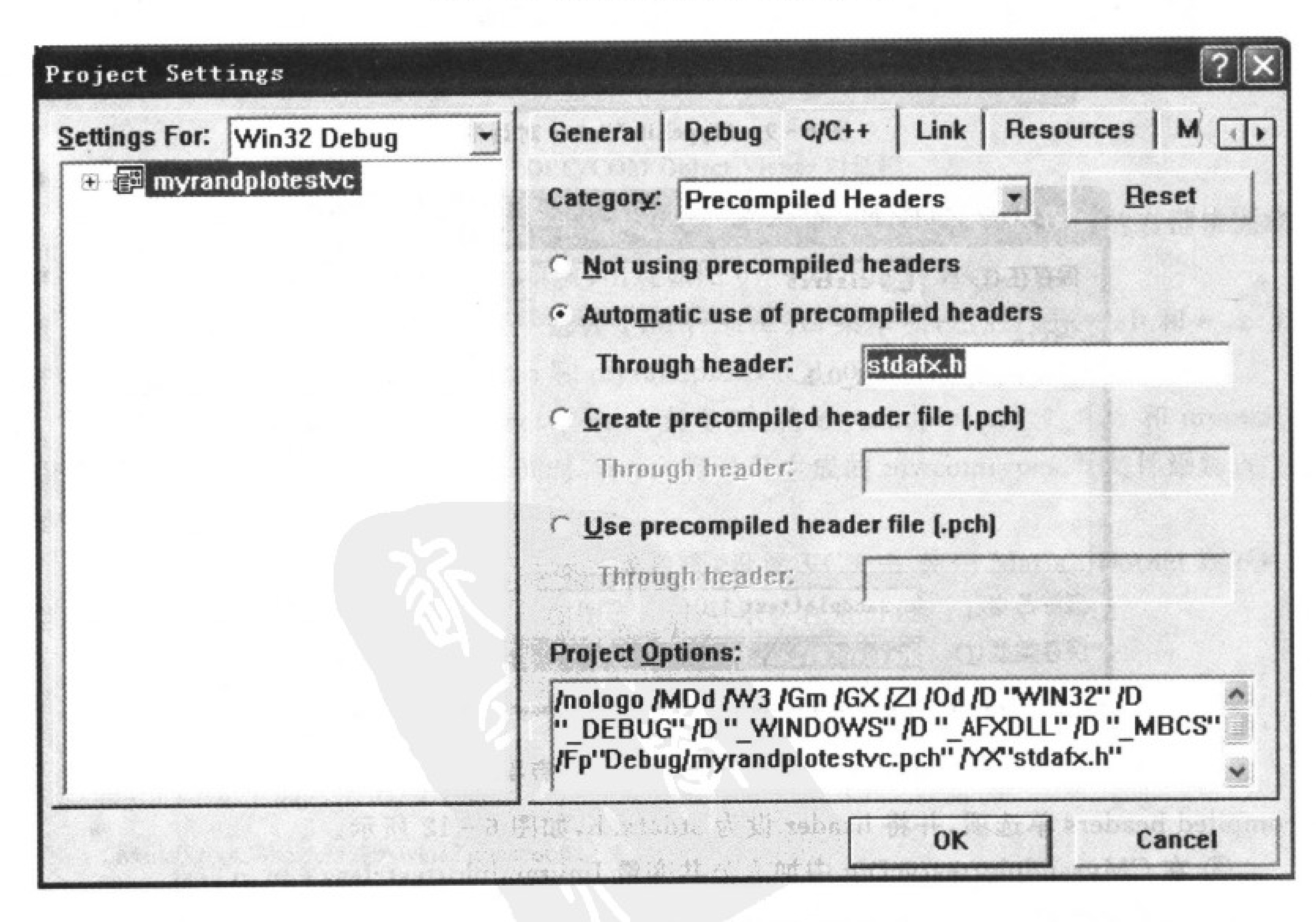


图 6-12　工程 C/C++选项卡设置

两个按钮,并通过 ClassWizard 添加上述两个按钮的消息响应函数。

完成后的 CMyrandplottestvcDlg 的源代码如下所示。

```
/* myrandplottestvcDlg.h 文件内容 */
#if ! defined(AFX_MYRANDPLOTTESTVCDLG_H__0E83166F_F438_4FA3_8FAF_8C7E9F6AC60E__INCLUDED_)
#define AFX_MYRANDPLOTTESTVCDLG_H__0E83166F_F438_4FA3_8FAF_8C7E9F6AC60E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CMyrandplottestvcDlg dialog

#include "mwcomutil.h"
#import "D:\Matlab6p5p1\bin\win32\mwcomutil.dll" raw_interfaces_only
#include "myrandplottest_1_0.h"

class CMyrandplottestvcDlg : public CDialog
{
// Construction
public:
    CMyrandplottestvcDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    //{{AFX_DATA(CMyrandplottestvcDlg)
    enum { IDD = IDD_MYRANDPLOTTESTVC_DIALOG };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMyrandplottestvcDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;
    Imyrandplottestclass * m_pTest;

    // Generated message map functions
    //{{AFX_MSG(CMyrandplottestvcDlg)
    virtual BOOL OnInitDialog();
```

```
    afx_msg void OnSysCommand(UINT nID,LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnTest();
    afx_msg void OnQuit();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft VC++ will insert additional declarations immediately before the previous line.

#endif // ! defined(AFX_MYRANDPLOTTESTVCDLG_H__0E83166F_F438_4FA3_8FAF_8C7E9F6AC60E__
INCLUDED_)

// myrandplottestvcDlg.cpp : implementation file
//

#include "stdafx.h"
#include "myrandplottestvc.h"
#include "myrandplottestvcDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
```

```
    protected:
    virtual void DoDataExchange(CDataExchange * pDX);      // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange * pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMyrandplottestvcDlg dialog

CMyrandplottestvcDlg::CMyrandplottestvcDlg(CWnd * pParent /* =NULL */)
    : CDialog(CMyrandplottestvcDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CMyrandplottestvcDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp() -> LoadIcon(IDR_MAINFRAME);
}
```

```
void CMyrandplottestvcDlg::DoDataExchange(CDataExchange * pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMyrandplottestvcDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMyrandplottestvcDlg,CDialog)
    //{{AFX_MSG_MAP(CMyrandplottestvcDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDTEST,OnTest)
    ON_BN_CLICKED(IDQUIT,OnQuit)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMyrandplottestvcDlg message handlers

BOOL CMyrandplottestvcDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu * pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
    {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING,IDM_ABOUTBOX,strAboutMenu);
    }
 }
    // Set the icon for this dialog.  The framework does this automatically
```

```
        .................................................// when the application's main window is not a dialog
    SetIcon(m_hIcon,TRUE);          // Set big icon
    SetIcon(m_hIcon,FALSE);         // Set small icon

    // TODO: Add extra initialization here
    m_pTest = NULL;
    if ((FAILED(CoInitialize(NULL))))
    {
        AfxMessageBox("初始化出错!",MB_OK,NULL);
    }
    return TRUE;   // return TRUE  unless you set the focus to a control
}

void CMyrandplottestvcDlg::OnSysCommand(UINT nID,LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID,lParam);
    }
}

// If you add a minimize button to your dialog,you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CMyrandplottestvcDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND,(WPARAM) dc.GetSafeHdc(),0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
```

```
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x,y,m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CMyrandplottestvcDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CMyrandplottestvcDlg::OnTest()
{
    // TODO: Add your control notification handler code here
    // Initialize COM
    HRESULT hr;
    Imyrandplottestclass * pTest = NULL;
    hr = CoCreateInstance(CLSID_myrandplottestclass,NULL,

    CLSCTX_ALL,IID_Imyrandplottestclass,(void ** )&pTest);
    if(FAILED(hr))
    {
        AfxMessageBox("创建 Imyrandplottestclass 出错");
        return;
    }

    COleVariant in = 100.0;
    SAFEARRAY * pa;
    double * pColor = NULL;
    pa = SafeArrayCreateVector(VT_R8,0,3);
    hr = SafeArrayAccessData(pa,(void ** )&pColor);
    if(FAILED(hr))
    {
        return;
    }
```

```
    /* color=[0 1 1];setrandplotcolor(color); */
    pColor[0]=0;
    pColor[1]=1.0;
    pColor[2]=1.0
    SafeArrayUnaccessData(pa);
    VARIANT color;
    color.vt=VT_R8|VT_ARRAY;
    color.parray=pa;
    pTest->setplotcolor(color);
    pTest->myrandplot(0,NULL,(VARIANT)in);
    m_pTest=pTest;
}

void CMyrandplottestvcDlg::OnQuit()
{
    // TODO: Add your control notification handler code here
    if(m_pTest! =NULL)
    {
        m_pTest->Release();
    m_pTest=NULL;
    }
    OnCancel();
    CoUninitialize();
}
```

myrandplottestvc 工程运行的结果如图 6-13 所示。

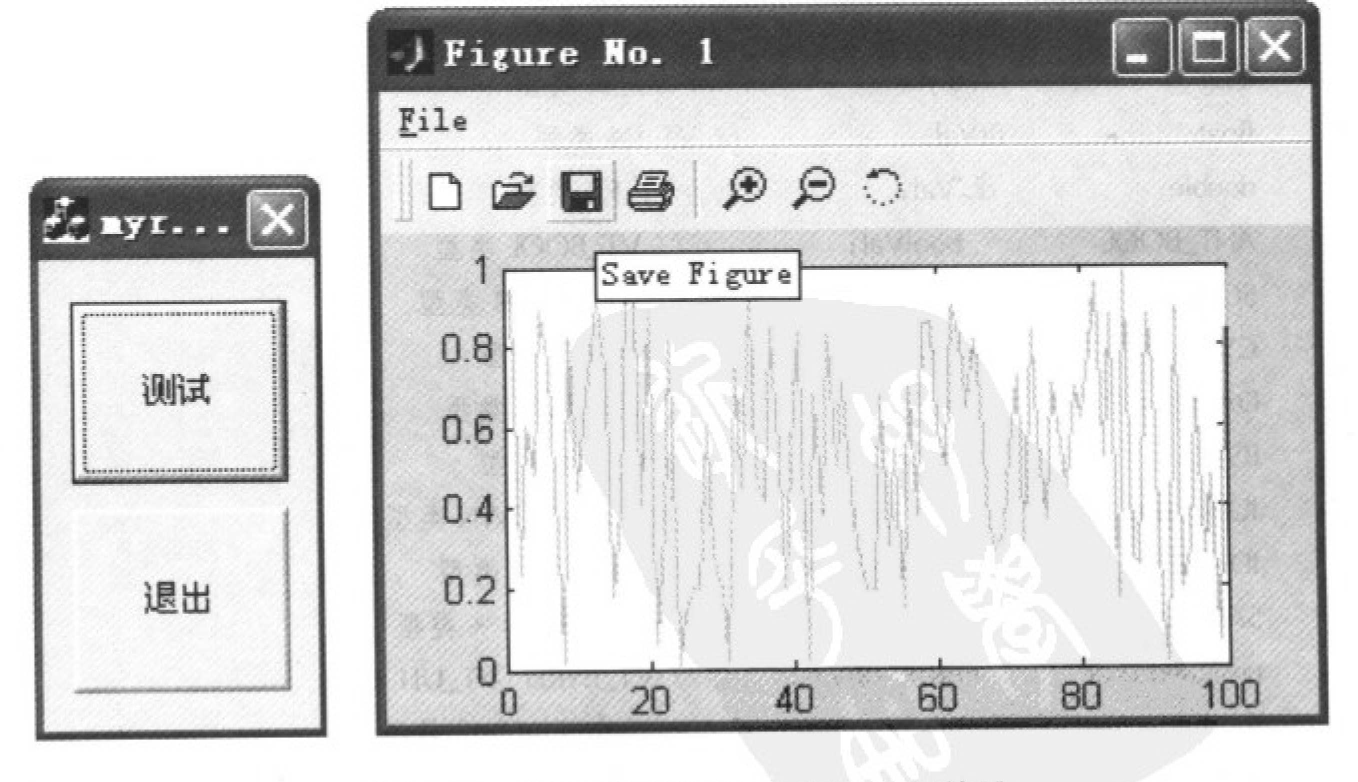


图 6-13　myrandplottestvc 工程的运行结果

6.4 Matlab Dotnet Builder 与 Visual C++之间的数据转换

若要在 VC++中调用 Matlab Dotnet Builder 生成的 COM 组件，那么客户和组件之间的数据交换是一个必须要解决的问题。采用 Matlab Dotnet Builder 生成的控件，COM 组件和客户之间的通过 VARIANT 数据类型交换数据。

6.4.1 VARIANT 数据类型

VARIANT 是 COM 组件与客户进行数据交互常用的数据类型，可以将其看做一种通用的数据类型。其实，在 C/C++语言中 VARIANT 是一个结构体。通过 VARIANT 数据类型，为 COM 组件与客户提供了一种非常有效的数据交互机制。因为它本身既包含了数据本身，也包含了数据的类型，当 COM 组件的开发语言与其客户的开发语言遇到类型转换问题的时候，可以利用 VARIANT 类型的这种特性实现不同语言之间的数据交互。下面即为 VARIANT 数据类型的定义。

```
typedef struct tagVARIANT
{
    VARTYPE vt;
    unsigned short wReserved1;
    unsigned short wReserved2;
    unsigned short wReserved3;
    union
    {
        unsigned char   bVal;               // VT_UI1 类型
        short           iVal;               // VT_I2 类型
        long            lVal;               // VT_I4 类型
        float           fltVal;             // VT_R4 类型
        double          dblVal;             // VT_R8 类型
        ANT_BOOL        boolVal;            // VT_BOOL 类型
        SCODE           scode;              // VT_ERROR 类型
        CY              cyVal;              // VT_CY 类型
        DATE            date;               // VT_DATE 类型
        BSTR            bstrVal;            // VT_BSTR 类型
        IUnknown        FAR * punkVal;      // VT_UNKNOWN 类型
        IDispatch       FAR * pdispVal;     // VT_DISPATCH 类型
        SAFEARRAY       FAR * parray;       // VT_ARRAY| * 类型
        unsigned char   FAR * pbVal;        // VT_BYREF|VT_UI1 类型
        short           FAR * piVal;        // VT_BYREF|VT_I2 类型
        long            FAR * plVal;        // VT_BYREF|VT_I4 类型
        float           FAR * pfltVal;      // VT_BYREF|VT_R4 类型
        double          FAR * pdblVal;      // VT_BYREF|VT_R8 类型
```

```
        VARIANT_BOOL       FAR * pboolVal;      // VT_BYREF|VT_BOOL 类型
        SCODE              FAR * pscode;        // VT_BYREF|VT_ERROR 类型
        CY                 FAR * pcyVal;        // VT_BYREF|VT_CY 类型
        DATE               FAR * pdate;         // VT_BYREF|VT_DATE 类型
        BSTR               FAR * pbstrVal;      // VT_BYREF|VT_BSTR 类型
        IUnknown FAR *     FAR * ppunkVal;      // VT_BYREF|VT_UNKNOWN 类型
        IDispatch FAR *    FAR * ppdispVal;     // VT_BYREF|VT_DISPATCH 类型
        SAFEARRAY FAR *    FAR * pparray;       // VT_ARRAY|* 类型
        VARIANT            FAR * pvarVal;       // VT_BYREF|VT_VARIANT 类型
        void               FAR * byref;         // Generic ByRef 类型
      };
  };
```

可以看出，VARIANT 类型除了保留字节以外，还包含了一个类型成员 vt 以及一个大的 union 类型。为了实现跨语言的特性，COM 接口在传递参数的时候采用这种 VARIANT 类型的参数，这样可以将函数参数的类型检查推迟到运行时刻来完成。

```
...
    IMyInterface * pInterface;
    ...
    VARIANT para;
    VariantInit(&para);
    para.vt = VT_I2;
    para.iVal = 10;
    pInterface->MyFun(para);
    ...
```

例如，在上述例子中，COM 对象在调用方法 MyFun 的时候并不知道 para 参数的类型，只是在函数真正要运行的时刻才能通过检查 VARIANT 变量的 vt 项来判断接收到的参数是 VT_I2 型(即 16 位整型)，然后再根据参数的类型取相应的值(iVal)。为了方便处理 VARIANT 类型的变量，Windows 提供了这样一些非常有用的函数：

- VariantInit　初始化 VARIANT 变量，将 vt 设置为 vt_EMPTY。
- VariantClear　清除现有的 VARIANT 变量，并将其初始化。
- VariantChangeType　改变 VARIANT 变量的类型。
- VariantCopy　复制 VARIANT 变量。

使用 VARIANT 变量时需要注意两点：

① VARIANT 变量使用前一定要调用 VariantInit 进行初始化。

② 有时候函数的有些参数是可选的，如果不想给可选的参数提供一个值，可以传递 VT_ERROR 给 vt 域，则与 vt 域对应的 scode 域则被设置为 DISP_E_PARAMNOTFOUND，相应的 COM 方法也应该提供他自己的默认值。

VARIANT 类型使用起来比较麻烦，使用 VC 进行程序设计的时候，还有另外两个封装好的 VARIANT 类可以使用，即 ColeVariant 和_variant_t。

ColeVariant 类的构造函数将 VARIANT 类型的初始化和类型转换隐藏起来，在对象构

造的时候,构造函数首先调用 VariantInit 进行初始化,然后根据参数中的标准类型调用相应的构造函数,并使用 VariantCopy 进行转换赋值操作,当 VARIANT 对象不在有效范围时,ColeVariant 对象就会自动析构,并由析构函数调用 VariantClear,因此不必考虑 VARIANT 类型内存的清除问题。ColeVariant 的赋值操作符在与 VARIANT 类型转换中也会为用户提供极大的方便。

例如,下面的代码:

```
//VT_I4,1234,32 位整型,值为 1234
ColeVariant varI4 = (long)1234;
//VT_R8,1234.0,double 型,值为 1234.0
ColeVariant varR8 = (double)1234.0;
//VT_BSTR,"Hello World",字符串型,值为"Hello World"
ColeVariant varBSTR = "Hello World!";
```

ColeVariant 主要用于自动化(Automation),而_variant_t 是一个用于 COM 开发的 VARIANT 类,它的功能与 ColeVariant 相似。

采用 ColeVariant 时需要头文件 Afxdisp. h,采用_variant_t 时需要头文件 COMDEF. H。

6.4.2 SAFEARRAY 数据类型

通过 SafeArray,可以在 VC++和 Matlab Dotnet Builder 之间传递数组参数。为了保证程序和 SafeArray 结构无关,COM 库提供了一套 API 函数用于处理 SafeArray,程序中建立、读取、更改和释放 SafeArray 都应该通过这些 API 进行,而不应该直接读写 SafeArray 结构。下面就来分析一下结构体 SafeArray 的定义,在 32 位 Windows 操作系统和 16 位 Windows 操作系统上,SafeArray 的定义稍有不同。

```
typedef struct FARSTRUCT tagSAFEARRAY
{
    unsigned short cDims;// 数组维的数目
    unsigned short fFeatures; // 数组释放方式标志
    #if defined(WIN32)
        unsigned long cbElements;// // 数组元素个数
        unsigned long cLocks;
    #else
        unsigned short cbElements;
        unsigned short cLocks;
        unsigned long handle;
    #endif
    void HUGEP * pvData;          //指向数组数据的指针
    SAFEARRAYBOUND rgsabound[1];  //数组各维边界
}SAFEARRAY;
```

SAFEARRAYBOUND 结构体用来描述数组的维数和数组各维索引的上下界,其定义如下所示。

```
typedef struct tagSAFEARRAYBOUND
{
    unsigned long cElements;
    long lLbound;
}SAFEARRAYBOUND;
```

其中域 cElements 代表数组某一维元素的个数，lLbound 代表数组某一维元素索引的下界。

如果要创建一个 3×2 的数组，每一维的下界都是 0，那么需要创建下面的 SAFEARRAYBOUND 数组：

```
...
SAFEARRAYBOUND bound[2];//数组一共两维，第一维用 bound[0]来描述，
                         //第二维用 bound[1]来描述。
    //要创建 3×2 的数组，初始化 bound[0]和 bound[1]
    bound[0].lLbound = 0;
    bound[1].lLbound = 0;
    bound[0].cElements = 3;
    bound[1].cElements = 2;
```

6.4.3 SAFEARRAY 的创建函数

1. SAFEARRAY * SafeArrayCreate(

VARTYPE vt,
unsigned int cDims,
SAFEARRRAYBOUND FAR * rgsabound
);

创建一个 SAFEARRAY 数组，其中 vt 为数组的类型，不能为 VT_ARRAY 和 VT_BYREF，而且 VT_EMPTY and VT_NULL 作为参数也是无效的；cDims 为数组的维数；rgsabound 为数组每一维索引的最小值和元素数目的描述。下面有两个例子，第一个用以说明如何创建一个 1 * L 的数组。第二个用以说明如何创建一个 m * n 的二维数组，多维数组的创建方法与此类似。

(1)如何创建 1 * L 的数组

```
//创建一个一维整型数组，长度为 L
SAFEARRAY FAR * psa;
SAFEARRAYBOUND rgsabound[1];
rgsabound[0].lLbound = 0;
rgsabound[0].cElements = L;
psa = SafeArrayCreate(VT_I4, 1, rgsabound);
...
```

(2)如何创建 m * n 的二准数组

```
//创建一个二维 double 型数组,数组的维数为:m * n
...
SAFEARRAY FAR * psa;
SAFEARRAYBOUND rgsabound[2];
rgsabound[0].lLbound = 0;
rgsabound[1].lLbound = 0;
rgsabound[0].cElements = m;
rgsabound[1].cElements = n;
psa = SafeArrayCreate(VT_R8,2,rgsabound);
...
```

2. SAFEARRAY * SafeArrayCreateVector(
VARTYPE vt,
long lLbound,
unsigned int cElements
);

利用 SafeArrayCreateVector 可以方便地创建一个一维数组,例如上面第一个创建 1×L 数组的例子可以用 SafeArrayCreateVector 实现。

```
...
//采用 SafeArratCreateVector 创建一个一维数组
SAFEARRAY FAR * psa;
psa = SafeArrayCreateVector(VT_I4,0,L);
...
```

6.4.4 Matlab Dotnet Builder 与 Visual C++数据转换

当 VC++调用由 Matlab Dotnet Builder 生成的 COM 组件时,一方面,在 VC++客户程序中需要用 VARIANT 类型来构造 Matlab Dotnet Builder COM 组件接口函数所需要的输入参数,另一方面,VC++客户程序中需要用 VARIANT 类型的变量来接收 Matlab Dotnet Builder COM 组件接口函数的输出结果。

1. Matlab 数据类型的 VARIANT 类型表示

(1) 数值阵列

Matlab 数值阵列根据所表示数据类型和精度的不同,可以分为 8 位、16 位、32 位有符号整型和无符号整型、单精度和双精度浮点类型等。所有这些数值阵列都可以通过 SAFEARRAY 及相应的"<任意 VARIANT 类型>| VT_ARRAY 类型"的 VARIANT 变量来表示。对于图 1×1 的数值阵列,可以按照上述方法表示。另外,也可以直接用非 VT_ARRAY 类型的 VARIANT 变量来表示。如对于如下的 Matlab 变量 a=5.0,可以采用下面的 VARIANT 变量形式来表示。

```
VARIANT va;
va.vt = VT_R8;
va.dblVal = 5.0;
```

(2) 字符阵列

Matlab 一维字符阵列可以采用 VT_BSTR 类型的 VARIANT 变量来实现。对于多维字符阵列，可以采用 VT_BSTR|VT_ARRAY 类型的 VARIANT 变量来实现。在 VC++中当客户调用 Matlab Dotnet Builder COM 组件时，如果用 VARIANT 变量表示的多维字符阵列作为输入参数，则需要注意多个字符串的长度必须一致。在这种情况下，最好采用元组的 VARIANT 形式传送字符串。

(3) 元组阵列

对于元组阵列，可以采用 VT_VARIANT|VT_ARRAY 类型的 VARIANT 变量来表示。VARIANT 变量实际上是一个 VARIANT 类型的数组，数组的任何一个元素是一个 VARIANT 变量，这个 VARIANT 变量可以表示 Matlab 的任一阵列类型，从而达到用 VARIANT 类型变量来表示元组阵列的目的。

2. VARIANT 变量和 Matlab 阵列的转换规则

(1) Matlab 阵列类型到 COM VARIANT 类型变量转换规则

Matlab 各种类型的阵列与 VARIANT 变量的转换规则如表 6-1 所列。

(2)COM VARIANT 类型变量与 Matlab 阵列类型的转换规则

VARIANT 变量与 Matlab 各种阵列类型的转换规则如表 6-2 所列。

表 6-1　Matlab Dotnet Builder 与 VARIANT 类型数据之间的转换类型对照表

Matlab 数据类型	标量数据对应的 VARIANT 类型	阵列数据对应的 VARIANT 类型
cell	对于 1×1 的 Matlab 元组，其对应的 VARIANT 类型与元组的内容有关，即其对应的 VARIANT 类型为元组内容对应的阵列类型按照转换规则转换后的 VARIANT 类型	对于 Matlab 元组阵列，其 VARIANT 类型为 VT_VARIANT|VT_ARRAY，即其对应的是一个 VARIANT 类型变量数组
struct	结构体对应 VT_DISPATCH 类型的 VARIANT 变量，Matlab 结构体被转换为 MWStruct 对象，这个对象通过 VT_DISPATCH 类型的 VARIANT 变量传递	与标量数据相同
char	对于 1×1 的字符类型阵列，其对应的 VARIANT 类型为 VT_BSTR，只是其字符串长度为 1	对于 1×L 的 Matlab 字符类型阵列，其对应的 VARIANT 类型为 VT_BSTR，且其字符串长度为 L 对于 M×L，M>1 甚至更高维的 Matlab 字符阵列，其对应的 VARIANT 类型为 VT_BSTR|VT_ARRAY，并且 VARIANT 字符串数组的每个元素的长度均为 1，与 Matlab 字符阵列的元素一一对应
sparse	Matlab 稀疏矩阵阵列对应 VT_DISPAATCH 类型的 VARIANT 变量，Matlab 结构体被转换为 MWSparse 对象，这个对象通过 VARIANT 变量传递	与标量数据相同

续表 6-1

Matlab 数据类型	标量数据对应的 VARIANT 类型	阵列数据对应的 VARIANT 类型
double	对于 1×1 的实数双精度浮点数值阵列，其对应的 VARIANT 类型为 VT_R8。对于 1×1 的复数双精度浮点数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数双精度浮点数值阵列被转换为 MWComplex 对象	对于大于 1×1 的实数双精度浮点 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_R8|VT_ARRAY。对于大于 1×1 的复数双精度浮点 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数双精度浮点数值阵列被转换为 MWComplex 对象
single	对于 1×1 的实数单精度浮点数值阵列，其对应的 VARIANT 类型为 VT_R4。对于 1×1 的复数单精度浮点数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数单精度浮点数值阵列被转换为 MWComplex 对象	对于大于 1×1 的实数单精度浮点 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_R4|VT_ARRAY。对于大于 1×1 的复数单精度浮点 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数单精度浮点数值阵列被转换为 MWComplex 对象
int8 uint8 int16 uint16 int32 uint32	对于 1×1 的实数(U)X 位整型数值阵列，其对应的 VARIANT 类型为 VT_(U)IX。对于 1×1 的复数 (U)X 位整型数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数(U)X 位整型数值阵列被转换为 MWComplex 对象	对于大于 1×1 的实数 (U)X 位整型 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_(U)IX|VT_ARRAY。对于大于 1×1 的复数(U)X 位整型 Matlab 数值阵列，其对应的 VARIANT 类型为 VT_DISPATCH，Matlab 复数(U)X 位整型数值阵列被转换为 MWComplex 对象
logical	对于 1×1 的逻辑型 Matlab 阵列，其对应的 VARIANT 类型为 VT_BOOL	对于大于 1×1 的逻辑型 Matalb 阵列，其对应的 VARIANT 类型为 VT_BOOL|VT_ARRAY

表 6-2　VARIANT 类型数据与 Matlab Dotnet Builder 之间的数据转换类型对照表

VARIANT type	Matlab 数据类型	说　明
VT_EMPTY	N/A	空阵列(empty)
VT_I1 VT_UI1 VT_I2 VT_UI2 VT_I4 VT_UI4	int 8 uint8 int16 uint16 int32 uint32	8、16、32 位有符号整型标量和无符号整型标量
VT_R4 VT_R8	single double	单精度和双精度浮点型变量
VT_CY	double	货币类型(currency)
VT_BSTR	char	1×L 的 VT_BSTR 类型的 VARIANT 变量转换为 Matlab 字符数组。将 VT_BSTR|VT_ARRAY 类型的 VARIANT 变量转换为 Matlab 元组(cell)，其元素为 1×L 的字符数组
VT_ERROR	int32	HRESULT 类型的变量，表示 COM 组件的错误状态

续表 6-2

VARIANT type	Matlab 数据类型	说　明
VT_DATE	double	同样是 double 型数据，在 Matlab 和 VARIANT 中，日期类型的数据起点不同，在 VARIANT 类型中，日期的起点是 1899 年 11 月 31 日；在 Matlab 中，日期的起点是 0/0/00 00:00:00，所以当把 VARIANT 的日期类型映射到 Matlab 中时，需要加上 693960.0，VARIANT 的日期类型可以转换为字符串
VT_INT VT_UINT	int unsigned int	无符号整型和有符号整型，分别与 int 和 unsigned int 类型对应
VT_DECIMAL	Double	
VT_BOOL	logical	IDispatch * 指针现在支持 Excel Range
VT_DISPATCH	可变类型	用以处理 Matlab 结构阵列、复数阵列、稀疏矩阵阵列等比较复杂类型的阵列，请读者参考 Matlab Dotnet Builder 关于使用工具库中 MWStruct，MWComplex，MWSparse 和 MWArg 等类使用方法的介绍
<anything>| VT_BYREF	可变类型	引用类型，任何基本类型都可引用，转换后的 Matlab 数组包含对数据的深复制(deep copy)
<anything>| VT_ARRAY	可变类型	多维 VARIANT 数组转换为多维 Matlab 数组，数组的类型与基本数据类型及其转换规则有关，多维 VT_VARIANT|VT_ARRAY 类型的 VARIANT 数组会被转换为多维元组(cell)，每个元组的转换方式与转换规则有关

3. 数组格式标志(array formatting flags)

Matlab Dotnet Builder 组件通过相应的标志(flags)控制数组数据在 Matlab Dotnet Builder 和调用 COM 组建的客户之间的数据交互方式。通常，可以在使用 COM 组件的客户中，通过设置相应的数组格式标志，以满足 COM 组件接口函数的需要。Matlab Dotnet Builder 组件的数组格式标志的类型为 MWArrayFormatFlag。表 6-3 给出了 Matlab 数组格式标志及其作用。

表 6-3　Matlab 数组格式标志及其作用

数组格式标志	作　用
InputArrayFormat	定义输入数组需要的数组格式化规则，输入数组是由客户端创建的 VARIANT 数组，作为 COM 对象接口的某个输入参数。此标志的有效值为：mwArrayFormatAsIs，mwArrayFormatMatrix，mwArrayFormatCell mwArrayFormatAsIs 表示输入数组不变 mwArrayFormatMatrix 表示将所有的输入数组当作矩阵。当输入 VARIANT 是类型 VT_ARRAY|<type>时，<type>是任何数值类型，此标志无效。当输入 VARIANT 是类型 VT_VARIANT|VT_ARRAY 时，则检查其中的 VARIANT 变量，如果它们都是单值而且是同一类型的话，此时采用相应类型的 Matlab 矩阵，而不采用元组 mwArrayFormatCell 表示将所有的输入数组当作元组(cell)
InputArrayIndFlag	当输入 Matlab 阵列为 VT_VARIANT|VT_ARRAY 类型时，InputArrayIndFlag 表示 InputArrayFormat 起作用的层数。默认情况下，InputArrayIndFlag 为 0，表示 Input|ArrayIndFlag 只对输入参数的第一层起作用
OutputArrayFormat	输出数组的转换规则，有效值为 mwArrayFormatAsIs，mwArrayFormatMatrix，mwArrayFormatCell
OutputArrayIndFlag	与 InputArrayIndFlag 类似，OutputArrayIndFlag 表示 OutputArrayFormat 起作用的层数
AutoResizeOutput	Excel ranges only
TransposeOutput	转置标志

4. 数据转换标志(data conversion flags)

Matlab Dotnet Builder 生成的 COM 组件通过数据转换标志来控制 VARIANT 类型与 Matlab 阵列类型转换时使用的规则。Matlab Dotnet Builder 生成的 COM 组件的数据转换标志的类型为 MWArrayFormatFlag。数据转换标志共包含 CoerceNumericToType、InputDateFormat、OutputAsDate 和 DateBias 四个属性。

(1) mwDataType CoerceNumericToType ——Matlab 数值阵列强制转换标志

CoerceNumericToType 标志通过 Matlab Dotnet Builder 生成的 COM 组件的数据转换器(data converter)将所有的 VARIANT 数值数据转换为特定的 Matlab 类型。受影响的数据类型包括 VT_I1, VT_UI1, VT_I2, VT_UI2, VT_I4, VT_UI4, VT_R4, VT_R8, VT_CY, VT_DECIAML, VT_INT, VT_UINT, VT_ERROR, VT_BOOL 和 VT_DATE。

mwDataType 的定义在 mwcomtypes.h 中可以找到，其定义如下。

```
typedef enum mwDataType
    {
        mwTypeDefault = 0, /* CoerceNumericToType 的默认值 */
        mwTypeLogical = 3,
        mwTypeChar    = 4,
        mwTypeDouble  = 6,
        mwTypeSingle  = 7,
        mwTypeInt8    = 8,
        mwTypeUint8   = 9,
        mwTypeInt16   = 10,
        mwTypeUint16  = 11,
        mwTypeInt32   = 12,
        mwTypeUint32  = 13
}mwDataType;
```

(2) mwDateFormat InputDateFormat ——输入日期类型

InputDateFormat 标志通知数据转换器，将 VARIANT 日期类型转换为 Matlab 日期类型。mwDateFormat 的定义也可以在 mwcomtypes.h 中找到，其定义如下。

```
typedef enum mwDateFormat
    {
        mwDateFormatNumeric = 0,
        mwDateFormatString  = 1
    }mwDateFormat;
```

其中，

mwDateFormatNumeric 作为默认值，表示输入的 VARIANT 日期类型按照图 6-2 的规则转换

mwDateFormatString 表示输入的 VARIANT 日期类型转换为字符串型。

(3) bool OutputAsDate ——日期输出标志

如果 OutputAsDate 为 TRUE ，则 Matlab Dotnet Builder COM 组件数据转换器将输出

作为日期类型；如果 OutputAsDate 为 FALSE，则 Matlab Dotnet Builder COM 组件数据转换器将输出作为双精度实型数据。

(4)DateBias As Long ——VARIANT 与 Matlab 日期差值

设置 VARIANT 与 Matlab 的日期转换的差值，其默认值为 693960，可以更改为任意一个 long 型整数。

6.5 Matlab COM 工具库

6.5.1 简 介

Matlab Dotnet Builder 包含一组非常方便的数据转换工具库，其中包括用于处理 Matlab 特有数据结构与 VC++客户交互的工具，如结构阵列(struct)、元组阵列(cell)和稀疏矩阵(sparse matrix)等。这个工具库通过 COM 的形式提供给 VC++用户，主要包含在 mwcomutil.dll 中，可以通过 DOS 下面的命令行注册：

```
mwregsvr mwcomutil.dll
```

可以通过下面的方法直接使用(<matlabroot>表示 Matlab 安装的根目录)：

```
# import "<matlabroot> \bin\win32\mwcomutil.dll" raw_interfaces_only
```

6.5.2 工具库的类(utility library classes)

Matlab Dotnet Builder 工具库包含下面 7 个主要的类：

- MWUtil 类；
- MWFlags 类；
- MWStruct 类；
- MWField 类；
- MWComplex 类；
- MWSparse 类；
- MWArg 类。

1. MWUtil 类

MWUtil 类包含一组在数组处理中非常有用的函数。对于 Dotnet Builder 客户而言，MWUtil 可以将 VARIANT 变量组成 VARIANT 数组，将 VARIANT 数组拆分为 VARIANT 变量以及 Dotnet Builder 与 VARIANT 日期类型之间的转换等功能，由于这里只是利用 MWUtil 提供的功能函数，所以在客户程序中最好存在一个全局范围内的 MWUtil 对象实例，这样可以提高客户访问 MWUtil 对象实例的效率。MWUtil 与 Dotnet Builder 相关的主要函数是：

① MWPack 用于将多个不同长度的 VARIANT 变量打包成一个 VARIANT 数组，典型的用途是将多个分散的输入打包成一个 varargin cell 的输入形式，最多将 32 个输入参数打包。

② MWUnpack 与 MWPack 相反，将一个 VARIANT 数组拆为多个独立的 VARIANT 变量，其典型用途是将输出的 varargout 数组打开成独立的变量。

③ MWDate2VariantDate(pVar)提供 Dotnet Builder 与 VARIANT 的日期格式转换。

2. MWFlags 类

MWFlags 类包含一组数组格式化和数据转换标志，所有的 Matlab Dotnet Builder 组件都包含一个 MWFlags 实例。通过修改 MWFlags 可以方便地改变对象修改数据转换规则。这个类包含下面的属性变量和函数：

- MWArrayFormatFlag ArrayFormatFlags　数组格式化标志；
- MWDataConversionFlags DataConversionFlags　数据转换标志；
- HRESULT Clone(IMWFlags ** ppFlags)；　复制当前 MWFlags 实例。

(1) MWArrayFormatFlags ArrayFormatFlags

ArrayFormatFlags 用于控制 Matlab Dotnet Builder 控件输入输出数据的转换规则。MWArrayFormatFlags 类是不可以创建的(noncreatable class)，只能通过某个已经存在的 MWFlags 的对象实例来获取 MWArrayFormatFlags 的访问权。MWArrayFormatFlags 类包含下面六个属性变量：

- mwArrayFormat InputArrayFormat；
- long InputArrayIndFlag；
- mwArrayFormat OutputArrayFormat；
- long OutputArrayIndFlag；
- VARIANT AutoResizeOutput；
- VARIANT TransposeOutput；

1) mwArrayFormat InputArrayFomrat

InputArrayFomrat 用来控制输入数组的格式，其默认值为 mwArrayFormatAsIs。可根据默认的转换规则(如表 6－1 所列)进行转换。

2) long InputArrayIndFlag

对于嵌套数组(例如，传递的 VARIANT 数组，其数组中每一个 VARIANT 元素是数组本身)，InputArrayIndFlag 表示 InputArrayFormat 适用的 VARIANT 数组层数，默认值为 0。

对于 Varargin 参数，不要更改 InputArrayIndFlag 的值，因为数据转换模块在遇到 Varargin 元组时会自动将 InputArrayIndFlag 加 1。

3) mwArrayFormat OutputArrayFormat

OutputArrayFormat 用来控制输出数组格式，其默认值为 mwArrayFormatAsIs 。此时可以根据默认的转换规则(如表 6－1 所列)进行转换。

4) long OutputArrayIndFlag

OutputArrayIndFlag 表示 OutputArrayIndFlag 适用的 VARIANT 数组层数，与 InputArrayIndFlag 类似，其默认值为 0。碰到 Varargout 参数时自动加 1。

5) VARIANT AutoResizeOutput

AutoResizeOutput 是 VT_BOOL 类型的 VARIANT 变量，这个属性只在与 Excel Builder 相关的应用中起作用。

6) VARIANT TransposeOutput

TransposeOutput 是 VT_BOOL 型的 VARIANT 变量，其作用在于控制输出数组是否转置。

MWDataConversionFlags　DataConversionFlags

DataConversionFlags 属性控制输入参数需要强制转换时的规则，MWDataConversion-Flags 是不可创建的类，只能通过 MWFlags 类的实例来访问。

MWDataConversionFlags 包含下面四个属性：

- mwDataType　CoerceNumericToType；
- mwDateFormat　InputDateFormat；
- VARIANT　OutputAsDate；
- long DateBias。

其中属性 CoerceNumericToType 的作用在于可以将所有的数值输入参数强制转换为一种特定的 Matlab 类型，当调用 COM 对象的客户端的各种数值格式与 Matlab COM 组件的需求不一样时，这个属性就会非常有用。比如，可以将 long，int 等其他各种不同类型数据格式转换为 double 型。

其他属性变量的含义参见 6.4.4 节关于数据转换标志的说明。

(3) Clone(MWFlags ** ppFlags)

复制当前 MWFlags 对象。

3. MWStruct 类

使用 MWStruct 类，可以使调用 Matlab Dotnet Builder 的客户程序方便地对 Matlab 结构体进行操作。其主要的接口函数如下所述。

(1) HRESULT Initialize(VARIANT varDims,VARIANT varFieldNames) ;

Initialize 根据输入参数分配一个结构体。

输入参数：

VARIANT varDims　结构体的维数数组。

VARIANT varFieldNames　每个结构体字段的名字。

说明：

创建时 MWStruct 对象只是 1×1 维的结构体阵列，而且不包含任何域。Initialize 函数初始化新创建的结构体。两个参数可以选择输入；如果两个都不输入的话，那么与之对应的结构体性质不发生变化。例如，如果输入参数 varDims 没有输入，则结构体维数不变。

(2) HRESULT get_Item (

```
                    VARIANT i0,
                    VARIANT i1,
                    VARIANT i2,
                    VARIANT i3,
                    ...
                    VARIANT i31,
                    struct IMWField ** ppField
                  );
```

通过索引来获取结构体相应的域，返回放在 IMWField ** ppField 里面。可以提供下面

几种形式的索引。

1）直接用字段名

只有 1×1 的结构体数组才能这么做，例如对某个结构体数组的“color”字段采用字段名的索引方式为[“color”]，其实例代码如下所示。

```
MWComUtil::IMWStructPtr pStruct;
MWComUtil::IMWField * pField=NULL;
//将此结构体初始化为 1*1 的数组
const OLECHAR * FieldNames[2]={L"color",L"shape"};
VARIANT varMissing;
VARIANT varDims;
VARIANT varFields;
VARIANT varFieldName;
VariantInit(&varDims);
VariantInit(&varFields);
VariantInit(&varMissing);
VariantInit(&varFieldName);
varMissing.vt=VT_ERROR;
varMissing.scode=DISP_E_PARAMNOTFOUND;//这两条语句创建的 VARIANT 表示没有变量输入
GetDimsArray(1,1,varDims);
GetFieldNamesArray(nFields,FieldNames,varFields);
pStruct.CreateInstance(__uuidof(MWComUtil::MWStruct));
pStruct->Initialize(varDims,varFields);
//下面对结构体的各个域进行赋值操作
...
//直接采用字段名获取相应的域
CString strName="color";
varFieldName.vt=VT_BSTR;
varFieldName.bstrVal=strName.AllocSysString();
pStruct->get_Item(varFieldName,varMissing,…,varMissing,&pField);
//下面可以对获取的域进行操作
VARIANT value;
VariantInit(&value);
value.vt=VT_R8;
value.dblVal=20;
pField->put_Value(value);
```

2）采用索引和字段名称

采用简化的索引表达形式为[2,1,"color"]，[2,1,"color"]与[2,"color"]等价，其实例代码如下所示。

```
MWComUtil::IMWStructPtr pStruct;
MWComUtil::IMWField * pField=NULL;
//创建并初始化 2*2 二维结构体数组，结构体的域名和域的个数与上述一致
```

```
...
//采用索引和字段名称获取相应的域
VARIANT varIndex[2];
VariantInit(varIndex);
VariantInit(varIndex + 1);
varIndex[0].vt = VT_I4;
varIndex[0].iVal = 2;
varIndex[1].vt = VT_I4;
varIndex[1].iVal = 1;
VARIANT varFieldName;
VariantInit(&varFieldName);
CString strName = "color";
varFieldName.vt = VT_BSTR;
varFieldName.bstrVal = strName.AllocSysString();
//下面两种获取域的方式完全等价,产生的结果也一样
pStruct->get_Item(varIndex[0],varFieldName,varMissing,…,varMissing,&pField);
pStruct->get_Item(varIndex[0],varIndex[1],varFieldName,varMissing,…,varMissing,&pField);
//对获取的域进行操作
...
```

对于结构体的索引,需要注意如下两点:

① 字段名必须是最后一个参数。

② 字段名是对字母大小写敏感的。

(3) HRESULT get_FieldNames(VARIANT * pvarFieldNames);

得到结构体数组域的名称,pvarFieldNames 包含一个字符串的元组。

(4) HRESULT get_Dims (VARIANT * pvarDims);

得到结构体数组的维数信息,pvarDims 包含一个一维整型数组。

(5) HRESULT get_NumberOfFields (long * pnFields);

得到域的数目。

(6) HRESULT get_NumberOfDims (long * pnDims);

得到结构体数组的维数。

(7) clone(IMWStruct ** ppStruct)

复制结构体数组。

4. MWField 类

MWField 类包含 MWStruct 的字段信息的引用。MWField 类是不可创建的,其常用接口函数如下所述。

(1) 操作域值的 get_value 和 put_value 接口

HRESULT get_Value (VARIANT * pvarValue);

HRESULT put_Value (VARIANT pvarValue);

(2) 采用 get_Name 得到域的名称

HRESULT get_Name (BSTR * pbstrName);

(3) 操作字段转换标志的接口

MWField 类的 MWFlags 属性存储一个 MWFlags 对象的引用,作为相应字段的数据转换标志。当涉及到 MWStruct 的使用时,MWStruct 自己的 MWFlags 会覆盖调用对象的 MWFlags 以保证 MWStruct 数据转换的正确性。

HRESULT get_MWFlags (IMWFlags ** ppFlags);

HRESULT put_MWFlags (IMWFlags * ppFlags);

HRESULT Clone (IMWField ** ppField);

5. MWComplex 类

使用 MWComplex 类,可以使调用 Matlab Dotnet Builder 的客户程序方便地对 Matlab 复数类型数值阵列进行操作。其主要的接口函数如所述。

(1) 实部和虚部进行写出和读入操作

HRESULT get_Real (VARIANT * pvarValue);

HRESULT put_Real (VARIANT pvarValue);

HRESULT get_Imag (VARIANT * pvarValue);

HRESULT put_Imag (VARIANT pvarValue);

(2) MWFlags 数据转换标志的读入和写出操作

HRESULT get_MWFlags (IMWFlags ** ppFlags) ;

HRESULT put_MWFlags (IMWFlags * ppFlags);

(3) 复制操作

HRESULT Clone (IMWComplex ** ppComplex);

6. MWSparse 类

使用 MWSparse 类,可以使调用 Matlab Dotnet Builder 的客户程序方便地对 Matlab 稀疏矩阵数值阵列进行操作。其主要的接口函数如下所述。

(1) 对稀疏矩阵存储非零元素值的数组进行操作的函数。

稀疏矩阵存储非零元素数组的类型可以是任何能够强制转换成 VARIANT 的类型,而且必须能够分解和强制转换成 double 或 VARIANT_BOOL 类型的数值矩阵。

HRESULT get_Array (VARIANT * pvarArray) ;

HRESULT put_Array (VARIANT pvarArray);

(2) 稀疏矩阵行数和列数的操作函数

稀疏矩阵的行数和列数都是非负的。如果行数为零时,实际行数为行数索引值(RowIndex)的最大值;如果列数为零时,实际列数为列数索引值(ColumnIndex)的最大值。

HRESULT get_NumRows (long * pnRows) ;

HRESULT put_NumRows (long pnRows) ;

HRESULT get_NumColumns (long * pnColumns) ;

HRESULT put_NumColumns (long pnColumns) ;

(3) 稀疏矩阵的行数和列数的索引值操作函数

行数索引值存储了所有非零值的数组元素的索引。这个属性的值可以是任何能够强制转化为 VARIANT 类型的数值类型,也可以是对象类型,但是必须能分解或者强制转换成 long

型数值矩阵。如果行数(NumRows)是非零的,并且如果任何一行索引大于行数(NumRows),就会产生一个无效索引的错误。如果行数索引(RowIndex)数组的元素个数与存储非零元素值的数组的元素个数不相符,则也会产生一个错误。对于列数索引而言,同样存在上述问题。

HRESULT get_RowIndex (VARIANT * pIndex);

HRESULT put_RowIndex (VARIANT pIndex);

HRESULT get_ColumnIndex (VARIANT * pIndex);

HRESULT put_ColumnIndex (VARIANT pIndex);

(4) 稀疏矩阵复制函数

HRESULT Clone (struct IMWSparse ** ppSparse);

7. MWArg 类

MWArg 类用来向一个编译好的类传递参数,通过数据转换标志来实现将一个参数传入的目的。

(1) VARIANT Value

表示要传递的参数。

(2) MWFlags MWFlags

数据转换标志,此标志覆盖调用对象或者方法的所有 MWFlags。

(3) clone(MWArg ** ppArg)

复制 MWArg 类。

6.5.3 Matlab Dotnet Builder 的枚举类型

Matlab Dotnet Builder 工具库提供三种类型的枚举类型,其定义如下所述。

1. mwArrayFomat

mwArrayFormat 用来表示输入输出数据转换格式的规则。

```
typedef enum mwArrayFormat
{
    mwArrayFormatAsIs     = 0,//保持原有的数据格式不变
    mwArrayFormatMatrix = 1,//只要能转换为矩阵,则转换为矩阵
    mwArrayFormatCell     = 2 //只要能转换为元组,则转换为元组
}mwArrayFormat;
```

2. mwDataType

mwDataType 是一个表示数据类型的常数集,其表示的数据类型如图 6-4 所列。

```
typedef enum mwDataType
{
    mwTypeDefault = 0,
    mwTypeLogical = 3,
    mwTypeChar    = 4,
    mwTypeDouble = 6,
    mwTypeSingle  = 7,
```

```
    mwTypeInt8     = 8,
    mwTypeUint8    = 9,
    mwTypeInt16    = 10,
    mwTypeUint16   = 11,
    mwTypeInt32    = 12,
    mwTypeUint32   = 13
}mwDataType;
```

表 6-4 mwDataType 表示 Matlab 数据类型

常 数	数 值	Matlab 类型
mwTypeDefault	0	N/A
mwTypeLogical	3	logical
mwTypeChar	4	char
mwTypeDouble	6	double
mwTypeSingle	7	single
mwTypeInt8	8	int8
mwTypeUint8	9	uint8
mwTypeInt16	10	int16
mwTypeUint16	11	uint16
mwTypeInt32	12	int32
mwTypeUint32	13	uint32

3. mwDateFormat

mwDateFormat 表示日期转换格式的常数集。

```
typedef enum mwDateFormat
{
    mwDateFormatNumeric  = 0,//日期数据采用数值格式表示
    mwDateFormatString   = 1 //日期数据采用字符串格式表示
}mwDateFormat;
```

6.5.4 安装和发布控件

采用 Matlab Dotnet Builder 将控件制作完成以后,可以采用 Dotnet Builder 的 GUI 工具将控件及其所需要的 Matlab 库文件一并打包,以方便安装。Malab Dotnet Builder 制作的控件的安装和发布如表 6-5 所列。

表 6-5 Matlab Dotnet Builder 制作的控件的安装和发布

文 件	文件的功能
_install. bat	安装程序要调用的批处理文件
〈componentname_projectversion〉. dll	需要注册的组件动态链接文件
〈componentname〉. ctf	组件运行所需的 ctf 文件

用 DotnetTool 打开需要打包的工程,选择 Component | Package Component 菜单项,然

后出现图 6-14 所示的打包工具对话框，使用打包工具可以很方便地生成一个经过压缩的可自动执行的组件安装程序(选择 Include MCR 选项)。组件安装将下面这些文件压缩成一个文件，以方便开发人员使用。Dotnet Builder 生成的安装文件的名字为〈componentname〉.exe，如需安装，在目标机器上运行即可。

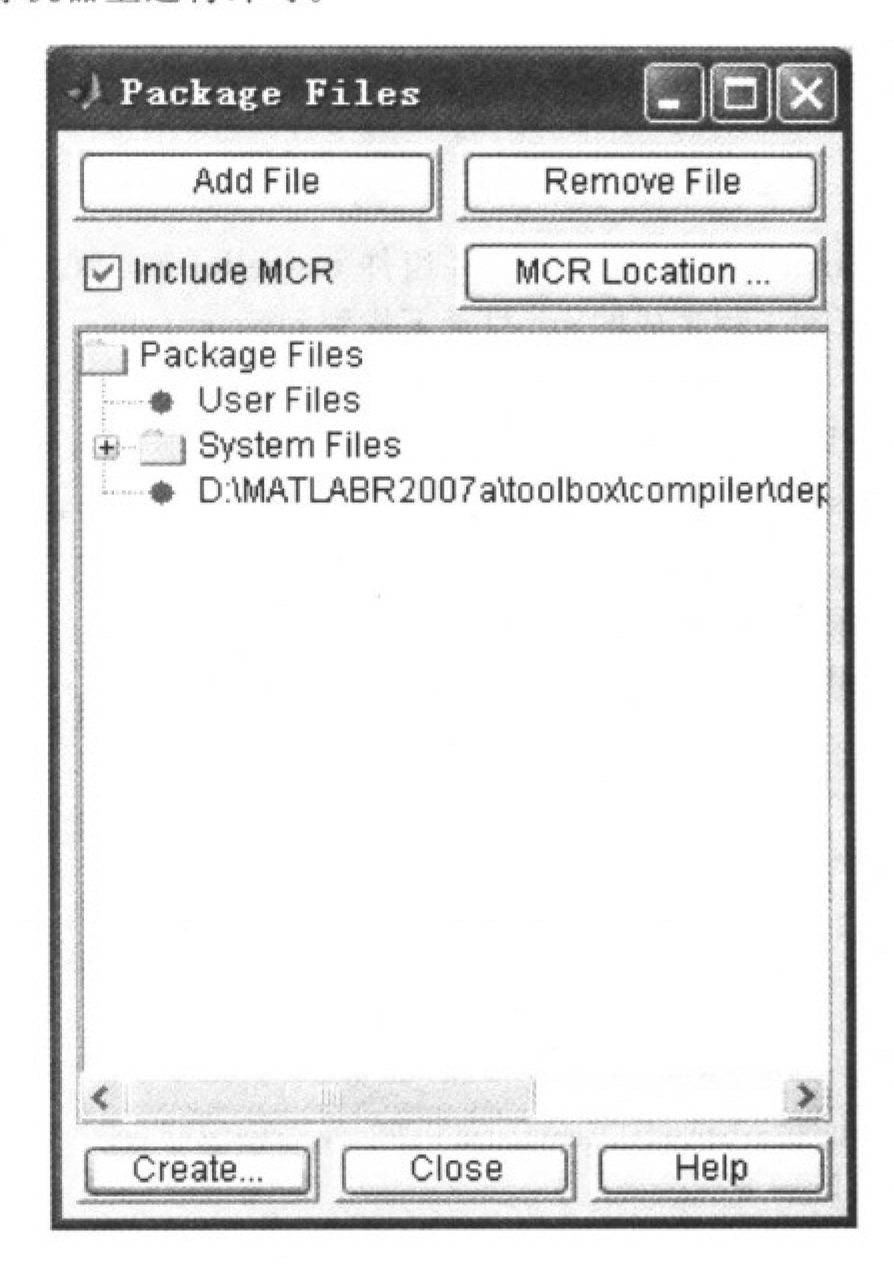


图 6-14　Matlab Dotnet Builder 安装文件打包对话框

6.6　综合实例

6.6.1　实例 1　数据转换及数组格式标志的使用

如果要调用 Matlab Dotnet Builder 生成的 COM 组件，不可避免地存在外部程序与 Matlab Dotnet Builder 生成的组件之间的数据交互问题，本实例向读者演示了如何通过 SAFEARRAY 向 Matlab Dotnet Builder 生成的组件传递数据，并说明如何设置数组格式标志。实例的主要功能是显示 COM 组件调用程序输入数据，其实现的步骤如下所述。

① 创建 M 文件 displyinput.m 的代码如下所示。

```
function [] = displayinput(x)
% function [] = displayinput(x)
```

```
%显示输入的数据
if isa(x,'cell')
    disp('The input data is a cell! ');
    x
else
    disp('The input data is a matrix! ');
    x
end
```

② 利用 Matlab Dotnet Builder 制作 COM 组件 testcellormatrix。

③ 在 testcellormatrix 工程生成的 src 目录下找到 mwcomtypes. h、testcellormatrix_idl. h 和 testcellormatrix_idl_i. c 文件(也可以利用 Visual C++ 6.0 Tools|OLE/COM Object Viewer 生成文件 testcellormatrix_1_0. h 和 testcellormatrix_1_0. c)。

④ 创建 Win32 console 工程 cellormatrixvc,将 mwcomtypes. h、testcellormatrix_idl. h、testcellormatrix_idl_i. c 文件加入到新建的 VC++工程中。

⑤ 编辑程序代码,编译执行程序。

其中主要代码在 cellormatrixvc. cpp 文件中实现,如下所示。

```
/* cellormatrixvc.cpp 文件内容 */
#include "stdafx.h"
#include <afxdisp.h>
#include "windows.h"
//#include "comutil.h"
//#include "mwutil.h"
#include "mwcomutil.h"
#import "D:\MATLABR2007a\bin\win32\mwcomutil.dll" raw_interfaces_only
#include "testcellormatrix_idl.h"
#include "COMDEF.H"
int main(int argc,char * argv[])
{
    // Initialize COM
    if ((FAILED(CoInitialize(NULL))))
    {
        printf("CoInitialize failed.\n");
    exit(1);
}

SAFEARRAY * pa;
pa = SafeArrayCreateVector(VT_VARIANT,0,3);
COleVariant var1 = "a";
COleVariant var2 = "d";
COleVariant var3 = "g";
VARIANT varIn;
```

```
VariantInit(&varIn);
varIn.vt = VT_VARIANT|VT_ARRAY;
VARIANT * data;
HRESULT hr;
hr = SafeArrayAccessData(pa,(void ** )&data);
if(FAILED(hr))
{
    return 0;
}
data[0] = var1;
data[1] = var2;
data[2] = var3;
SafeArrayUnaccessData(pa);
data = NULL;

varIn.parray = pa;

Itestcellormatrix * pCellOrMatrix = NULL;
hr = CoCreateInstance(CLSID_testcellormatrix,NULL,CLSCTX_ALL,IID_Itestcellormatrix,(void ** )&pCellOrMatrix);
if(FAILED(hr))
{
    return 0;
}

printf(" *********** Default Data Transformation ****************** \n");
printf("The InputArrayFormat is mwArrayFormatMatrix\n");
pCellOrMatrix->displayinput(varIn);

MWComUtil::IMWFlagsPtr pFlags = NULL;
MWComUtil::IMWArrayFormatFlagsPtr pArrayFormatFlags = NULL;
pFlags.CreateInstance(__uuidof(MWComUtil::MWFlags));
pFlags->get_ArrayFormatFlags(&pArrayFormatFlags);

mwArrayFormat arrayFormat;
//arrayFormat = mwArrayFormatMatrix;
arrayFormat = mwArrayFormatAsIs;

pArrayFormatFlags->put_InputArrayFormat(arrayFormat);

IMWFlags * pIFlags = NULL;
pFlags->QueryInterface(&pIFlags);

pCellOrMatrix->put_MWFlags(pIFlags);
```

```
printf(" ****** ************** Changed Data Transformation ************ \n");
printf("The InputArrayFormat is mwArrayFormatAsIs\n");
pCellOrMatrix->displayinput(varIn);


pCellOrMatrix->Release();
//pCellOrMatrix = NULL;

//VariantInit(&varIn);
varIn.vt = VT_EMPTY;
varIn.parray = NULL;
//SafeArrayDestroyData(pa);
pa = NULL;
//pa = NULL;
//data = NULL;
pArrayFormatFlags = NULL;
pFlags = NULL;
CoUninitialize();
return 0;
}
```

6.6.2 实例 2 采用 MWUtil 处理 varargin 输入和 varargout 输出

在 Matlab 程序设计中,通过 varargin 输入和 varargout 输出对不定个数的输入和输出参数进行处理。由于 Matlab Dotnet Builder 生成的组件实际上是用 Matlab 语言编写的,因而同样存在处理不定个数输入和输出参数的问题。在 C/C++语言中对输入和输出参数的个数要求必须是确定的,但是可以通过 Matlab 提供的 MWUtil 组件实现对 varargin 输入和 varargout 输出的处理。

实例 2 的实现步骤与实例 1 类似,首先创建三个函数文件 sortinputstring. m、createdictionary. m 和 displayinput. m。其中 sortinputstring 函数用来将输入的 varargin 元组阵列的字符串元素按照升序排序,createdictionary 用来创建本实例需要的 varargout 元组阵列,displayinput 用来显示 Matlab 阵列内容。然后利用 Matlab Dotnet Builder 将 sortinputstring. m,createdictionary. m 和 displayinput. m 编译为 COM 组件,并在 VC++ 6.0 客户程序中调用生成好的 COM 组件。

sortinputstring,createdictionary,displayinput 三个函数及 VC++ 6.0 的客户程序代码如下所示。

```
function [varargout] = sortinputstring(varargin)
% function [varargout] = sortinputstring[varargin]
% sortinputstring.m
varargout = sort(varargin);

function [varargout] = createdictionary()
```

```
    % function [y] = createdictionary()
    % createdictionary.m
    dic = cell(1,10);
    dic{1} = 'edit';
    dic{2} = 'file';
    dic{3} = 'view';
    dic{4} = 'window';
    dic{5} = 'help';
    dic{6} = 'format';
    dic{7} = 'project';
    dic{8} = 'search';
    dic{9} = 'column';
    dic{10} = 'properity';
    for i = 1:nargout
        varargout{i} = dic{i};
    end

    function [] = displayinput(x)
    % function [] = displayinput(x)
    % displayinput.m
    disp(x);

/* mwutilvc.cpp 文件内容 */
#include "stdafx.h"
#include "stdio.h"


#include <afxdisp.h>
#include "windows.h"
#include "mwcomutil.h"
#import "D:\MATLABR2007a\bin\win32\mwcomutil.dll" raw_interfaces_only
//#include "testcellormatrix_1_0.h"
#include "testmwutil_idl.h"
#include "COMDEF.H"

#define _TEST_STR_NUM 10

int main(int argc,char * argv[])
{
    if ((FAILED(CoInitialize(NULL))))
    {
        printf("CoInitialize failed.\n");
        exit(1);
```

```
}
HRESULT hr;
MWComUtil::IMWUtilPtr pIMWUtil = NULL;
hr = pIMWUtil.CreateInstance(__uuidof(MWComUtil::MWUtil));
if(FAILED(hr))
{
    return 0;
}

VARIANT varStr[_TEST_STR_NUM];
VARIANT varStrEmpty,varStrOut;
VariantInit(&varStrEmpty);
VariantInit(&varStrOut);
int i=0;
for(i=0;i<_TEST_STR_NUM;i++)
{
    VariantInit(varStr + i);
}
//OLECHAR tmpChar[1] = 'a';

Itestmwutil * pTestUtil = NULL;
hr = CoCreateInstance(CLSID_testmwutil,NULL,CLSCTX_ALL,IID_Itestmwutil,(void ** )&pTestUtil);

if(FAILED(hr))
{
    return 0;
}

//创建字典数据———元组 varargout
pTestUtil->createdictionary(_TEST_STR_NUM,&varStrOut);
printf("creating dictionary cell|:\n");
pTestUtil->displayinput(varStrOut);

VARIANT_BOOL isResize = 1;

printf("\n");
printf("UnPacking the created dictionary cell:\n");
pIMWUtil->MWUnpack(varStrOut,0,isResize,varStr,varStr + 1,varStr + 2,
                    varStr + 3,varStr + 4,varStr + 5,varStr + 6,varStr + 7,
                  varStr + 8,varStr + 9,NULL,NULL,NULL,NULL,
                  NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
                  NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
for(i=0;i<_TEST_STR_NUM;i++)
```

```
{
    pTestUtil->displayinput(varStr[i]);
}
pTestUtil->sortinputstring(10,&varStrOut,varStrOut);

printf("\n");
printf("sorting the dictionary cell:\n");
pTestUtil->displayinput(varStrOut);


pIMWUtil->MWUnpack(varStrOut,0,isResize,varStr,varStr+1,varStr+2,
                   varStr+3,varStr+4,varStr+5,varStr+6,varStr+7,
                    varStr+8,varStr+9,NULL,NULL,NULL,NULL,
                    NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,
                    NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL);
printf("\n");
printf("UnPacking the sorted dictionary cell:\n");
for(i=0;i<_TEST_STR_NUM;i++)
{
    pTestUtil->displayinput(varStr[i]);
}

pIMWUtil->Release();
pIMWUtil=NULL;
pTestUtil->Release();
pTestUtil=NULL;

CoUninitialize();
printf("\n\npress any key to exit\n");
getchar();
return 0;
}
```

6.6.3 实例 3 MWStruct 和 MWField 操作实例

Matlab 结构体阵列是 Matlab 的一类重要的数据类型。Matlab 结构体阵列远比一般意义上的 C/C++语言的结构体数组复杂的多，关于 Matlab 结构体阵列读者可以参考本书 1.2.1 节和 3.11 节关于 Matlab 结构体阵列的描述。在 Matlab Dotnet Builder 生成的 COM 组件的调用程序中，可以通过 Matlab 提供的 MWStruct 和 MWField 组件对 Matlab 结构体及 Matlab 结构体的域进行操作。

在本实例的代码中，添加了一个函数 ShowCOMError 用于显示 VC++ 6.0 在调用 Dotnet Buidler 生成的组件时的错误信息，读者不必深究其中的代码含义，如果有兴趣的话可以参

考相关介绍 COM 组件的书籍。

本实例的实现步骤与实例 1 有一点不同：本实例建立了两个 COM 组件 structcombinecom 和 structutils，并在 VC++中同时调用。structcombine. m、displaystruct. m 和实现 COM 组件调用的 mwstruct_test. cpp 代码如下所示。

```
function [structOut] = structcombine(structIn1,structIn2,name1,name2)
% function [structOut] = structcombine(structIn1,structIn2,name1,name2)
% 结构体输入输出类型转换
% 将输入的两个结构体重新组合成新的结构体
structOut = struct(name1,structIn1,name2,structIn2);

function [] = displaystruct(in)
% function [] = displaystruct(in)
disp(in);

/* mwstruct_test.cpp 文件内容 */
#include <stdio.h>
#import "D:\MATLABR2007a\bin\win32\mwcomutil.dll" raw_interfaces_only

#include "mwcomutil.h"
#include "structcombinecom_idl.h"
#include "structutils_idl.h"

//得到 COM 调用过程中的错误并向标准输入设备输出错误信息
void ShowCOMError(IUnknown * pUnk,REFIID iid)
{
    ISupportErrorInfo * pSupportErrorInfo = NULL;
    IErrorInfo * pErrorInfo = NULL;
    HRESULT hr = S_OK;
    wchar_t szTemp[128];
    _bstr_t bstrMessage = _bstr_t("");

    if (pUnk == NULL)
        return;
    if (SUCCEEDED(pUnk->QueryInterface(IID_ISupportErrorInfo,
                                        (void **)&pSupportErrorInfo)))
 {
    if (pSupportErrorInfo->InterfaceSupportsErrorInfo(iid) == S_OK)
    {
        GetErrorInfo(0,&pErrorInfo);
        BSTR bstrDesc;
        BSTR bstrSource;
        pErrorInfo->GetDescription(&bstrDesc);
```

```
            pErrorInfo->GetSource(&bstrSource);
            swprintf(szTemp,L"Error: 0x%08x",hr);
            bstrMessage += szTemp;
            bstrMessage += " Source: ";
            bstrMessage += _bstr_t(bstrSource);
            bstrMessage += " Description: ";
            bstrMessage += _bstr_t(bstrDesc);
            printf("%s\n",(char *)bstrMessage);
            if (bstrDesc != NULL)
                SysFreeString(bstrDesc);
            if (bstrSource != NULL)
                SysFreeString(bstrSource);
            pErrorInfo->Release();
            pErrorInfo=NULL;
            }
            pSupportErrorInfo->Release();
            pSupportErrorInfo=NULL;
        }
}

//创建一维 VT_I4 类型的 VARIANT 数组,其维数位 mxn
HRESULT GetDimsArray(int m,int n,VARIANT& var)
{
    int * pv=NULL;

    if (m < 1 || n < 1)
{
        printf("Invalid dimensions\n");
        return E_FAIL;
    }
    SAFEARRAYBOUND rgsabound[1]={{2,1}};
    SAFEARRAY * psa=SafeArrayCreate(VT_I4,1,rgsabound);
    if (psa==NULL)
    {
        printf("Error creating safearray\n");
        return E_FAIL;
    }
    if (FAILED(SafeArrayAccessData(psa,(void **)&pv)))
    {
        SafeArrayDestroy(psa);
        printf("Error accessing safearray\n");
        return E_FAIL;
    }
```

```
    pv[0] = m;
    pv[1] = n;
    SafeArrayUnaccessData(psa);
    var.vt = VT_I4|VT_ARRAY;
    var.parray = psa;
    return S_OK;
}

//创建一维 VT_BSTR 类型的 VARIANT 数组,并根据输入结构体域名 FieldNames 设置 VARIANT 数组的内容
HRESULT GetFieldNamesArray(int nFields,const OLECHAR ** FieldNames,VARIANT& var)
{
    BSTR * pv = NULL;

    if (nFields < 0)
    {
        printf("Invalid number of fields\n");
        return E_FAIL;
    }
    SAFEARRAYBOUND rgsabound[1] = {{nFields,1}};
    SAFEARRAY * psa = SafeArrayCreate(VT_BSTR,1,rgsabound);
    if (psa == NULL)
    {
        printf("Error creating safearray\n");
        return E_FAIL;
    }
    if (FAILED(SafeArrayAccessData(psa,(void **)&pv)))
    {
        SafeArrayDestroy(psa);
        printf("Error accessing safearray\n");
        return E_FAIL;
    }
    for (int i=0; i<nFields; i++)
    {
        pv[i] = SysAllocString(FieldNames[i]);
    }
    SafeArrayUnaccessData(psa);
    var.vt = VT_BSTR|VT_ARRAY;
    var.parray = psa;
    return S_OK;
}

// 创建 1×1 的结构体阵列,其内容根据输入 Values 设置,其域名根据输入 FieldNames 设置
// 输入参数如 FieldNames 和 Values 一一对应,即结构体域名位 FieldNames[i]的域值等于 Values[i],且
```

```
// 其个数都为 nFields。假定输入的 MWStruct * pStruct 已经初始化。
HRESULT CreateStruct(MWComUtil::IMWStructPtr& pStruct, int nFields, const OLECHAR ** FieldNames, const
VARIANT * values)
{

    MWComUtil::IMWField * pField = NULL;
    HRESULT hr = S_OK;
    VARIANT varMissing;
    VARIANT varDims;
    VARIANT varFields;
    VARIANT varFieldName;
    int i = 0;

    // 初始化临时变量
    VariantInit(&varDims);
    VariantInit(&varFields);
    VariantInit(&varMissing);
    VariantInit(&varFieldName);
    varMissing.vt = VT_ERROR;
    varMissing.scode = DISP_E_PARAMNOTFOUND;

    // 创建维数数组
    if (FAILED((hr = GetDimsArray(1, 1, varDims))))
        goto EXIT;
    // 创建域名数组
    if (FAILED((hr = GetFieldNamesArray(nFields, FieldNames, varFields))))
        goto EXIT;
    hr = pStruct->Initialize(varDims, varFields);

    if (FAILED(hr))
    {
        printf("Initialize struct failed.\n");
        goto EXIT;
    }
    for (i = 0; i<nFields; i++)
    {
        varFieldName.vt = VT_BSTR;
        varFieldName.bstrVal = SysAllocString(FieldNames[i]);
        hr = pStruct->get_Item(
varFieldName,
varMissing, varMissing, varMissing, varMissing, varMissing,
varMissing, varMissing, varMissing, varMissing, varMissing,
varMissing, varMissing, varMissing, varMissing, varMissing,
```

```
                                        varMissing, varMissing, varMissing, varMissing, varMissing,
varMissing, varMissing, varMissing, varMissing, varMissing,
varMissing, varMissing, varMissing, varMissing, varMissing,
varMissing, &pField);
            pField->put_Value(values[i]);
        pField->Release();
        pField=NULL;
        VariantClear(&varFieldName);
    }
EXIT:
    // 退出时释放内存
    VariantClear(&varDims);
    VariantClear(&varFields);
    VariantClear(&varFields);
    VariantClear(&varFieldName);
    if (pField != NULL)
        pField->Release();
    return hr;
}

// DoTest 函数创建一个由三个域的结构体,其域名分别位"a","b","c"
/*
    每个域的域值位一个 1×1 的双精度数值阵列,然后调用 Matlab Dotnet Builder 生成的 COM 组件的
    displaystruct 方法显示生成的结构体。
*/
HRESULT DoTest()
{
    HRESULT hr=S_OK;

    MWComUtil::IMWStructPtr pStruct;
    MWComUtil::IMWStructPtr pStruct1;

    const OLECHAR * FieldNames[3]={L"a",L"b",L"c"};
    VARIANT values[3];
    VARIANT varStruct;
    int nret=0;

    // 初始化 VARIANT 变量
    VariantInit(&varStruct);
    for (int i=0; i<3; i++)
    {
        VariantInit(&values[i]);
        values[i].vt=VT_R8;
```

```
        values[i].dblVal = (double)(i+1);
    }

    // 创建 1×1 结构体,其域分别为"a","b","c",相应的域值位 s.a=1,s.b=2,s.c=3

    hr = pStruct.CreateInstance(__uuidof(MWComUtil::MWStruct));
    pStruct1.CreateInstance(__uuidof(MWComUtil::MWStruct));

    if (FAILED(hr))
    {
        printf("CreateInstance on MWStruct failed.\n");
        hr = E_FAIL;
        goto EXIT;
    }
    hr = CreateStruct(pStruct,3,FieldNames,values);
    if (FAILED(hr))
    {
        printf("Creation of struct failed.\n");
        hr = E_FAIL;
        goto EXIT;
    }
    hr = pStruct->QueryInterface(IID_IDispatch,(void **)&(varStruct.pdispVal));
    if (FAILED(hr))
    {
        printf("QI failed for some reason??? \n");
        hr = E_FAIL;
        goto EXIT;
    }
    varStruct.vt = VT_DISPATCH;

EXIT:
    //退出并释放内存
    VariantClear(&varStruct);
    if (pStruct != NULL)
        pStruct = NULL;

    return hr;
}

//本实例的 main 函数
int main(int argc,const char ** argv)
{
    int n=0;
```

```
int nret = 0;

//初始化 COM
if ((FAILED(CoInitialize(NULL))))
{
    printf(" CoInitialize failed. \n");
    exit(1);
}

//调用 DoTest 测试结构体的创建
nret = (DoTest() == S_OK ? 0 : 1);

//下面的代码演示如何创建两个结构体,并通过调用 combinestruct 方法实现结构体合并
Istructcombinecom * pStructCombineCom = NULL;
HRESULT hr;
hr = CoCreateInstance(CLSID_structcombinecom, NULL,
CLSCTX_ALL, IID_Istructcombinecom, (void ** )&pStructCombineCom);
if (FAILED(hr))
{
    printf(" Struct CombineCom failed. \n");
    exit(1);
}

Istructutils * pStructUtils = NULL;
HRESULT hr1;
hr1 = CoCreateInstance(CLSID_structutils, NULL,
    CLSCTX_ALL, IID_Istructutils, (void ** )&pStructUtils);
if (FAILED(hr1))
{
    printf(" Struct Utils failed. \n");
    exit(1);
}


MWComUtil::IMWStructPtr pStruct1;
MWComUtil::IMWStructPtr pStruct2;

VARIANT varName1, varName2, varStruct1, varStruct2, varStructOut;
VariantInit(&varName1);
VariantInit(&varName2);
VariantInit(&varStruct1);
VariantInit(&varStruct2);
VariantInit(&varStructOut);
```

```
varName1.vt = VT_BSTR;
varName2.vt = VT_BSTR;
varName1.bstrVal = SysAllocString(L"struct1");
varName2.bstrVal = SysAllocString(L"struct2");

VARIANT values[2];


//初始化 VARIANT 变量
for (int i=0; i<2; i++)
{
    VariantInit(&values[i]);
    values[i].vt = VT_R8;
    values[i].dblVal = (double)(i+1);
}

const OLECHAR * FieldNames1[2] = {L"color",L"shape"};
const OLECHAR * FieldNames2[2] = {L"width",L"length"};



hr = pStruct1.CreateInstance(__uuidof(MWComUtil::MWStruct));
hr = pStruct2.CreateInstance(__uuidof(MWComUtil::MWStruct));

printf("ALL OVER\n");
CreateStruct(pStruct1,2,FieldNames1,values);
CreateStruct(pStruct2,2,FieldNames2,values);


pStruct1->QueryInterface(IID_IDispatch,(void **)&varStruct1.pdispVal);
pStruct2->QueryInterface(IID_IDispatch,(void **)&varStruct2.pdispVal);


varStruct1.vt = VT_DISPATCH;
varStruct2.vt = VT_DISPATCH;

pStructCombineCom->structcombine(1,&varStructOut,varStruct1,varStruct2,varName1,varName2);


pStructUtils->displaystruct(varStructOut);

printf("ALL OVER\n");
```

```
    VariantClear(&varStructOut);







    pStruct1 = NULL;
    pStruct2 = NULL;
    pStructUtils = NULL;


    //释放 COM
    CoUninitialize();
    return nret;
}
```

6.6.4 实例 4 MWComplex 操作实例

复型数值阵列在 Matlab 程序设计中是一种使用非常频繁的数据类型，Matlab 提供的 MWComplex 组件可以使开发人员在调用 Matlab Dotnet Builder 生成的组件的程序中操作复型数值阵列的数据。在下面的实例中，构造了两个输入复型数值阵列，通过调用 Matlab Dotnet Builder 生成的组件计算其和，然后显示两个输入的复型数值阵列和输出计算结果。读者熟悉了本实例的实现思路，任何其他关于复型数值阵列的操作都是与其类似的。

本实例的实现步骤与实例 1 相同，构造 Matlab COM 组件 M 文件包括 addcomplexarray. m 和 displayarray. m，实现 COM 组件调用的代码文件为 arraycomplex. cpp。本实例的程序代码如下所示。

```
function [y] = addcomplexarray(x1,x2)
% function [y] = addcomplexarray(x1,x2)
y = x1 + x2;

function [] = displayarray(x)
% function [] = displayarray(x)
disp(x);
```

```
/* arraycomplex.cpp 文件内容 */
#include "stdafx.h"
#include <afxdisp.h>
#include "stdio.h"
#include "mwcomutil.h"
#import "D:\MATLABR2007a\bin\win32\mwcomutil.dll" raw_interfaces_only
#include "arraypasscom_idl.h"
#include "COMDEF.H"
```

```
int main(int argc,char * argv[])
{
    // 初始化 COM
    if ((FAILED(CoInitialize(NULL))))
    {
        printf(" CoInitialize failed.\n");
        exit(1);
    }

    MWComUtil::IMWComplexPtr pComplexIn1;
    MWComUtil::IMWComplexPtr pComplexIn2;
    MWComUtil::IMWStructPtr pStruct;
    HRESULT hr,hr1,hr2;
    hr1 = pComplexIn1.CreateInstance(__uuidof(MWComUtil::MWComplex));
    hr2 = pComplexIn2.CreateInstance(__uuidof(MWComUtil::MWComplex));

    if((FAILED(hr1)||FAILED(hr2)))
    {
        printf("输入复数型数组创建失败！\n");
        return FALSE;
    }
    VARIANT varReal[2],varImag[2];
    VARIANT varOutComplex;
    VARIANT varComplexIn1,varComplexIn2;
    VariantInit(&varOutComplex);
    VariantInit(&varComplexIn1);
    VariantInit(&varComplexIn1);
    int i=0,j=0;
    for(i=0;i<2;i++)
    {
        VariantInit(&varReal[i]);
        varReal[i].vt = VT_R8|VT_ARRAY;
        VariantInit(&varImag[i]);
        varImag[i].vt = VT_R8|VT_ARRAY;
    }

    SAFEARRAY * pa[4];
    SAFEARRAYBOUND bound={4,0};
    double * data=NULL;
    for(i=0;i<4;i++)
    {
        pa[i] = SafeArrayCreate(VT_R8,1,&bound);
```

```
        SafeArrayAccessData(pa[i],(void ** )&data);
        for(j=0;j<4;j++)
        {
            data[j]=j+i;
        }
        SafeArrayUnaccessData(pa[i]);
    }

    varReal[0].parray=pa[0];
    varReal[1].parray=pa[2];
    varImag[0].parray=pa[3];
    varImag[1].parray=pa[4];
    Iarraypasscom * pArrayPasscom=NULL;
    hr=CoCreateInstance(CLSID_arraypasscom,NULL,CLSCTX_ALL,IID_Iarraypasscom,(void ** )&pArrayPasscom);
    if(FAILED(hr))
    {
        printf(" Iarraypasscom 创建失败\n");
        return 0;
    }

    pComplexIn1->put_Real(varReal[0]);
    pComplexIn1->put_Imag(varImag[0]);

    pComplexIn2->put_Real(varReal[0]);
    pComplexIn2->put_Imag(varImag[0]);

    varComplexIn1.vt=VT_DISPATCH;
    varComplexIn2.vt=VT_DISPATCH;
    pComplexIn1->QueryInterface(IID_IDispatch,(void ** )&varComplexIn1.pdispVal);
    pComplexIn2->QueryInterface(IID_IDispatch,(void ** )&varComplexIn2.pdispVal);

    pArrayPasscom->addcomplexarray(1,&varOutComplex,varComplexIn1,varComplexIn2);

    printf("输入复数数组 1\n");
    pArrayPasscom->displayarray(varComplexIn1);
    printf("输入复数数组 2\n");
    pArrayPasscom->displayarray(varComplexIn2);
    printf("相加后的输出复数数组 1\n");
    pArrayPasscom->displayarray(varOutComplex);
```

```
VariantInit(&varOutComplex);
VariantInit(&varComplexIn1);
VariantInit(&varComplexIn1);
for(i=0;i<2;i++)
{
    VariantInit(&varReal[i]);
    VariantInit(&varImag[i]);
}

SafeArrayDestroy(pa[0]);
SafeArrayDestroy(pa[1]);
SafeArrayDestroy(pa[2]);
SafeArrayDestroy(pa[3]);

pComplexIn1->Release();
pComplexIn1=NULL;
pComplexIn2->Release();
pComplexIn2=NULL;

pa[0]=NULL;
pa[1]=NULL;
pa[2]=NULL;
pa[3]=NULL;

//释放 COM 组件
CoUninitialize();
return 0;
}
```

6.6.5　实例 5　MWSParse 操作实例

当数值阵列的非零元素个数所占的比例很小时，采用稀疏矩阵阵列可以大大节省存储空间(关于稀疏矩阵请参考 3.9 节关于稀疏矩阵的说明)。Matlab 同样也提供了 MWSparse 组件以方便 Matlab Dotnet Builder 组件调用程序操作稀疏矩阵。本实例旨在向读者演示采用 MWSparse 操作稀疏矩阵阵列的方法。在下面的实现过程中，首先创建一个数值阵列，然后调用 sparsematrix 方法将其转换为稀疏矩阵阵列，再采用 MWSparse 得到稀疏矩阵的各属性，并将其结果输出到屏幕上。用于创建 COM 组件的 change2sparse. m 和 displaymatrix. m 文件以及调用 COM 组件的 SparseMatrixVc. cpp 文件内容如下所示。

```
function [y]=change2sparse(x)
    % function [y]=change2sparse(x)
    [i,j,x1]=find(x);
    [m,n]=size(x);
```

```
    y=sparse(i,j,x1,m,n);

    function []=displaymatrix(x)
    % function []=displaymatrix(x)
    disp(x);

/* SparseMatrixVc.cpp 文件内容 */
#include "stdafx.h"
#include <afxdisp.h>
#include "stdio.h"
#include "mwcomutil.h"
#import "D:\MATLABR2007a\bin\win32\mwcomutil.dll" raw_interfaces_only
#include "sparsematrix_idl.h"
#include "COMDEF.H"



int main(int argc,char * argv[])
{
    // 初始化 COM
    if ((FAILED(CoInitialize(NULL))))
    {
        printf("CoInitialize failed.\n");
        exit(1);
    }
    MWComUtil::IMWSparsePtr pSparse=NULL;
    HRESULT hr;


    //构造一个非零元素较少的稀疏矩阵
    SAFEARRAY * pa=NULL;
    SAFEARRAYBOUND bound[2];
    bound[0].cElements=10;
    bound[0].lLbound=0;
    bound[1].cElements=10;
    bound[1].lLbound=0;
    pa=SafeArrayCreate(VT_R8,2,bound);
    double * data;

    hr=SafeArrayAccessData(pa,(void ** )&data);
    if(FAILED(hr))
    {
        return 0;
```

```
}

int i=0;

for(i=0;i<9;i=i+2)
{
    data[i * 10 + i]=i+1;
}
SafeArrayUnaccessData(pa);


Isparsematrixclass * pSparseMatrix=NULL;
hr=
CoCreateInstance(CLSID_sparsematrixclass,NULL,CLSCTX_ALL,IID_Isparsematrixclass,(void ** )&pSparseMatrix);
if(FAILED(hr))
{
    return 0;
}

VARIANT varInput,varSparse,RowIndex,ColIndex;
VariantInit(&varInput);
varInput.vt=VT_R8|VT_ARRAY;
varInput.parray=pa;
VariantInit(&varSparse);
VariantInit(&RowIndex);
VariantInit(&ColIndex);
printf("Dislay The Original Matrix:\n");
pSparseMatrix->displaymatrix(varInput);
pSparseMatrix->sparsematrix(1,&varSparse,varInput);
printf("Display The Changed SparseMatrix:\n");
pSparseMatrix->displaymatrix(varSparse);

pSparse=varSparse.pdispVal;
pSparse->get_RowIndex(&ColIndex);
pSparse->get_ColumnIndex(&RowIndex);

VariantClear(&varSparse);
VariantClear(&varInput);

printf("The nozero element index:\n");
printf("row index:     ");
pSparseMatrix->displaymatrix(RowIndex);
printf("column index:");
```

```
    pSparseMatrix->displaymatrix(ColIndex);

    pSparseMatrix->Release();
    pSparseMatrix=NULL;
    pSparse=NULL;
    pa=NULL;

    //释放 COM 组件
    CoUninitialize();
    return 0;
}
```



第 7 章　Matcom 与 C/C++

Matcom 是 Mathtools 开发的世界上最早的 Matlab 到 C++的编译器。后来，由于 Mathtools 被 Matlab 公司收购，Matcom 对 Matlab 5.3 以后的版本就没有推出新的版本，其中 M 文件的编译及与 C/C++语言的接口都已经在 Matlab 6.0 以后的版本中实现。但是，对于采用 C++的工程和研究人员来说，Matcom 的 C++矩阵库(Matrix Lib)依然有一定的利用价值。Matcom 的 C++矩阵库提供双精度级的计算精度，其矩阵类型可以为复数、实数、稀疏矩阵及 n 维矩阵。Matcom C++矩阵库中包含了约六百个经过严格测试的函数，这些函数涉及线性代数、多项式数学、信号处理和文件 I/O 等方面。为了方便使用 C++的开发人员，Matcom C++矩阵库还重载了常用的运算符，例如矩阵的+、-、*、/运算符以及矩阵的索引操作符。另外，Matcom 也提供了很多类似 Matlab 函数的图形库函数，为用户开发曲线显示、图像显示及其他数据显示软件提供了一个更好的选择。

7.1　安装 Matcom

Matcom 的安装与普通 Windows 程序的安装过程类似，首先找到 Matcom 的安装文件，双击后出现如图 7-1 所示的安装启动界面。

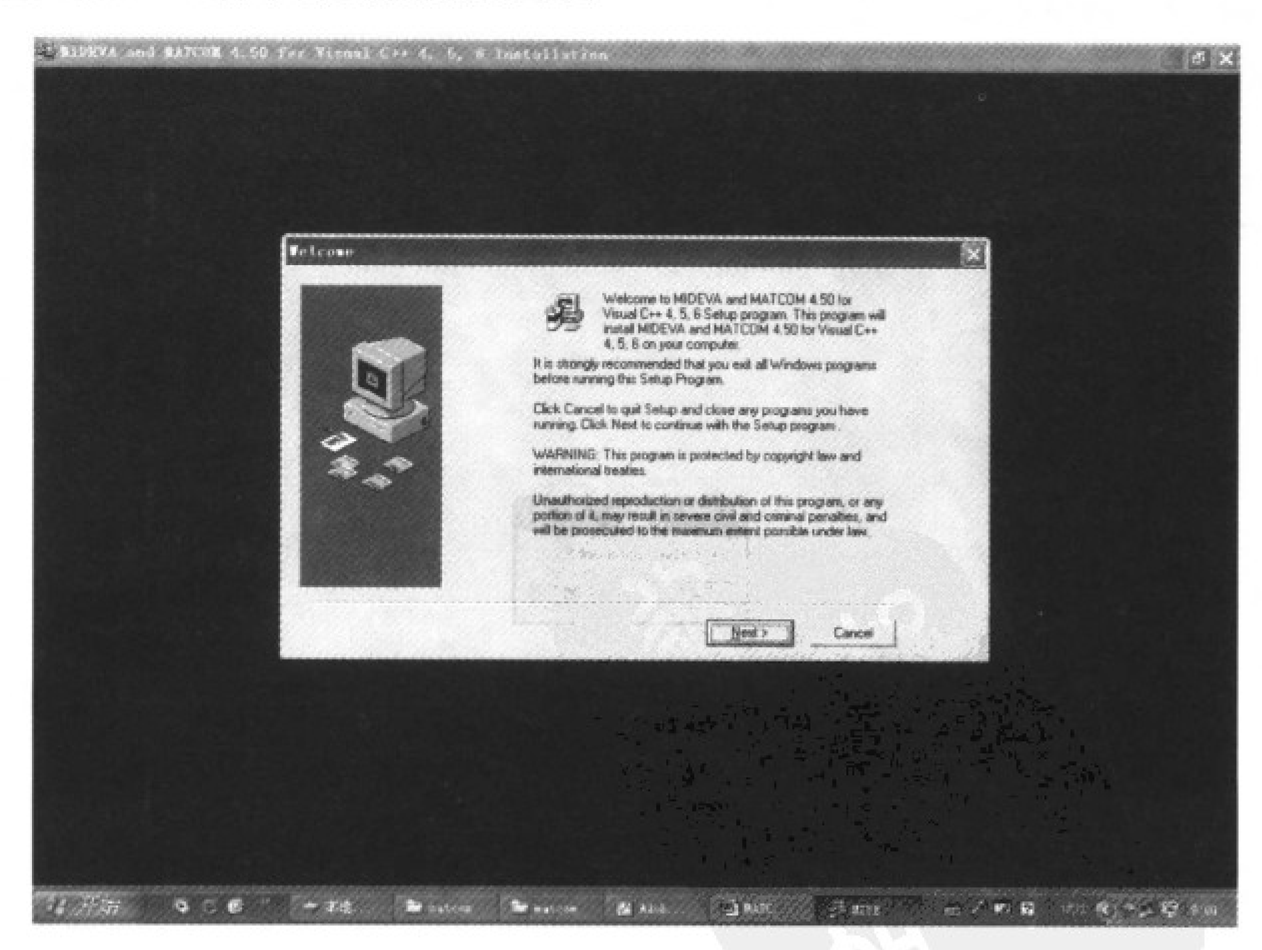


图 7-1　Matcom 安装界面——启动

按照安装程序的提示，逐步完成 Matcom 的中间安装过程。在安装快结束时，Matcom 安装程序会自动搜索本机上的 C/C++编译器，如 Visual C++等。本书所有的C/C++ 编译器

都是 Visual C++ 6.0。Matcom 搜索到 Visual C++ 6.0 的编译器后,出现如图 7-2 所示的提示,确认即可。

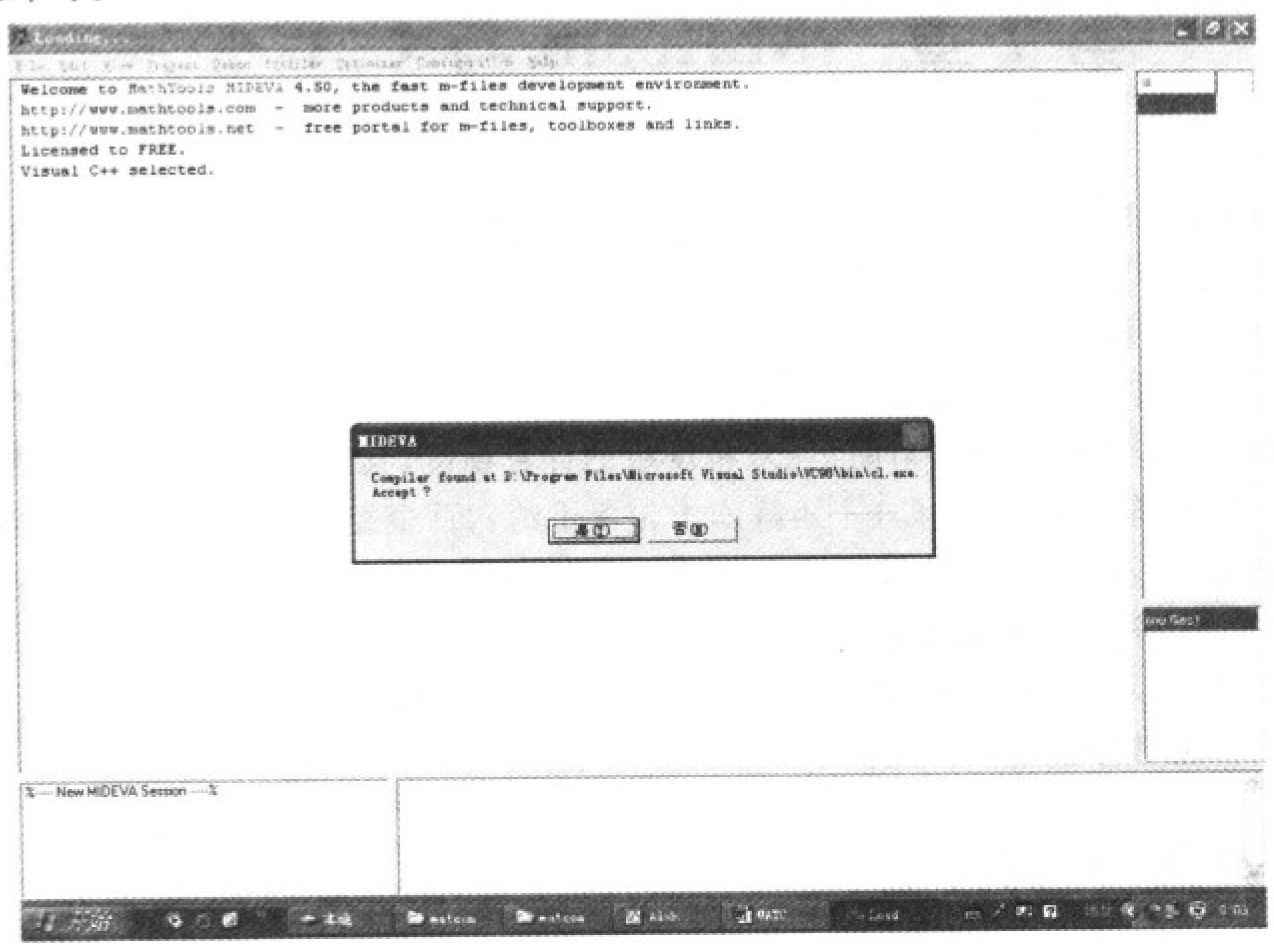


图 7-2 Matcom 安装界面——设置 C/C++编译器

接着 Matcom 询问本机是否存在 Matlab 安装版本,并弹出如图 7-3 所示的提示对话框,

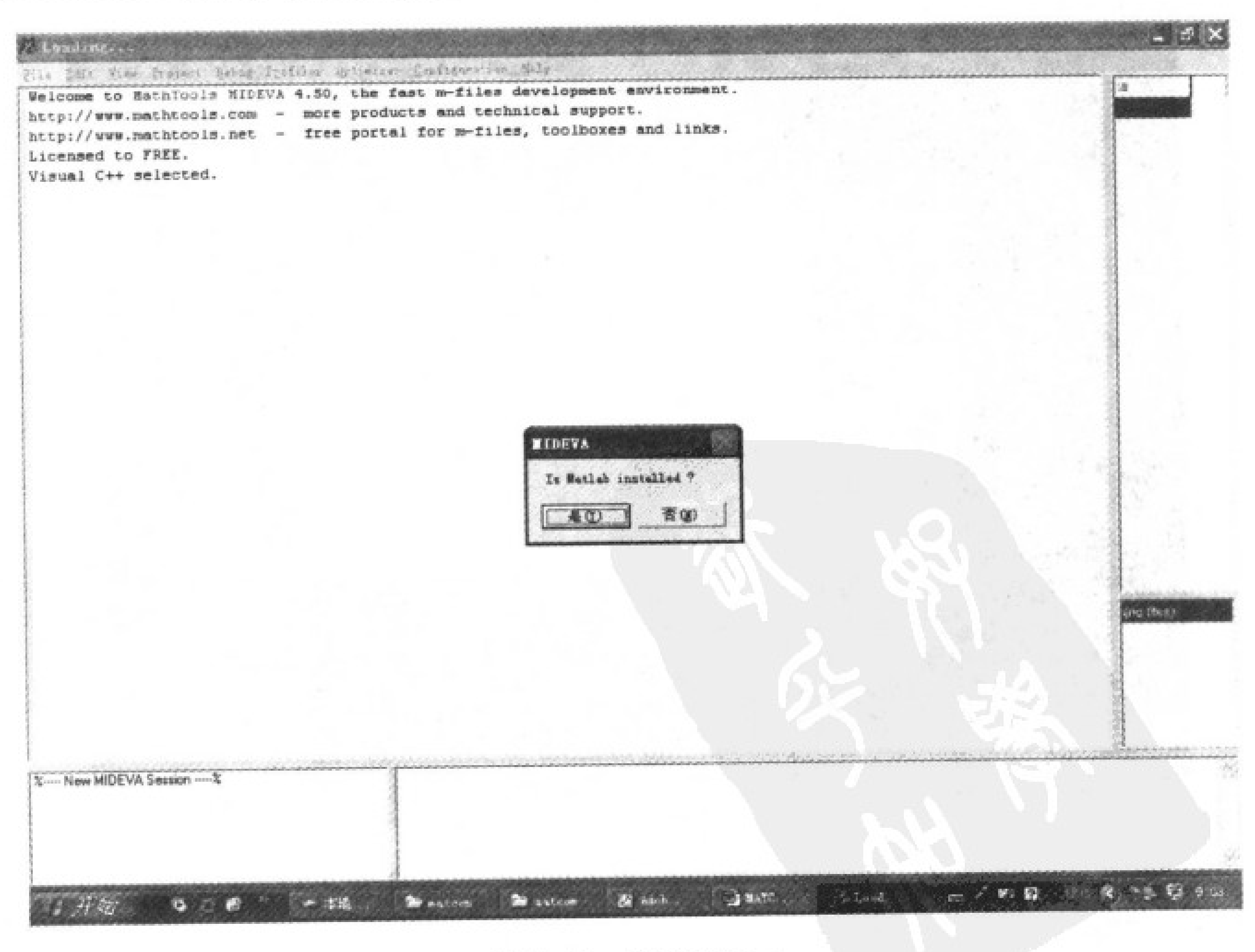


图 7-3 关联 Matlab

如果本机上安装的是 Matlab 5.3 版本，可以选择“是(Y)”确认，如果是其他版本的 Matlab，选“否(N)”。因为 Matcom 不支持 Matlab 6.1 及其以上的版本。但无论怎么选，都不会影响使用 Matcom C++矩阵库。

Matcom 成功安装完以后，在〈Matcom 根目录〉\lib\目录下可以找到使用 Matcom C++矩阵库的头文件 matlib.h 和 v4501v.lib 文件，在 Windows 操作系统的system32 目录下，可以找到使用 Matcom C++矩阵库的动态链接库文件 v4501v.dll。

7.2　在 VC++中使用 Matcom C++矩阵库

下面通过一个例子来说明 Matcom C++矩阵库在 Visual C++ 6.0 中的使用过程，对于其他的 C++编译器如 Borland C++，其使用过程类似。本实例的主要步骤如下：

- 建立一个新的工程，设置 Visual C++ 工程。
- 在需要使用 Matcom C++矩阵库的地方须加#include "matlib.h"说明。
- 初始化 Matcom C++矩阵库。
- 创建一个矩阵。
- 给矩阵元素赋值。
- 调用矩阵的函数。
- 得到矩阵的元素。

具体的操作步骤如下所述。

① 打开 Visual C++ 6.0，建立一个新的 Win32 Console 工程，如图 7-4 所示。设置 Visual C++工程，选择 Project|Setting 菜单项，在弹出的 Project Setting 对话框的 Link 选项

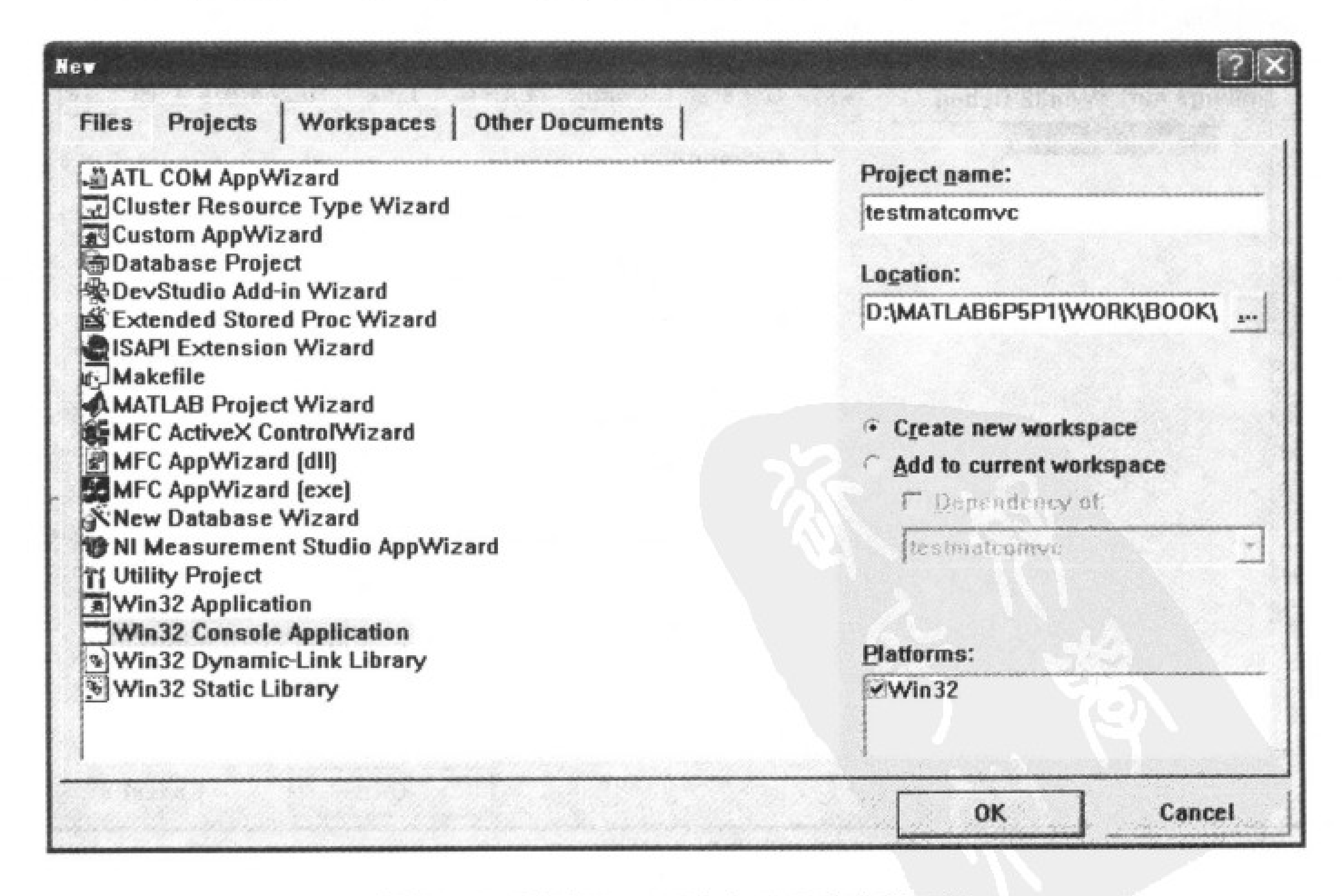


图 7-4　采用 Visual C++ 6.0 建立新工程

卡中，在 Category 中的 Input 选项下，加入矩阵库需要使用的 v4501v. lib，并且根据 Matcom 安装目录设置路径 c:\matcom45\lib。如图 7-5 所示。打开 C/C++选项卡，根据 Matcom 安装目录为头文件添加目录 c:\matcom45\lib。如图7-6 所示。

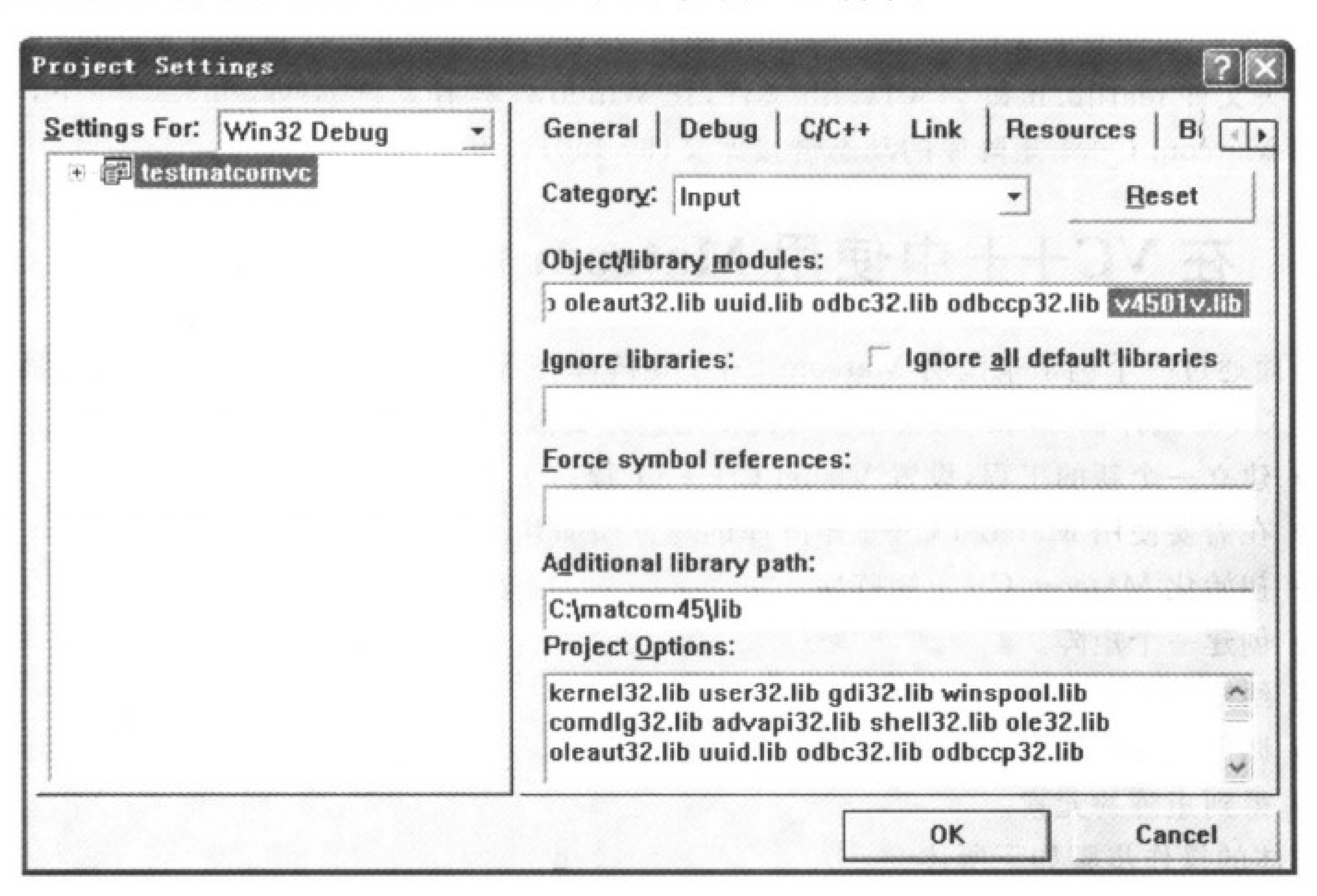


图 7-5 设置 Visual C++ 6.0 工程对话框(Link 选项卡)

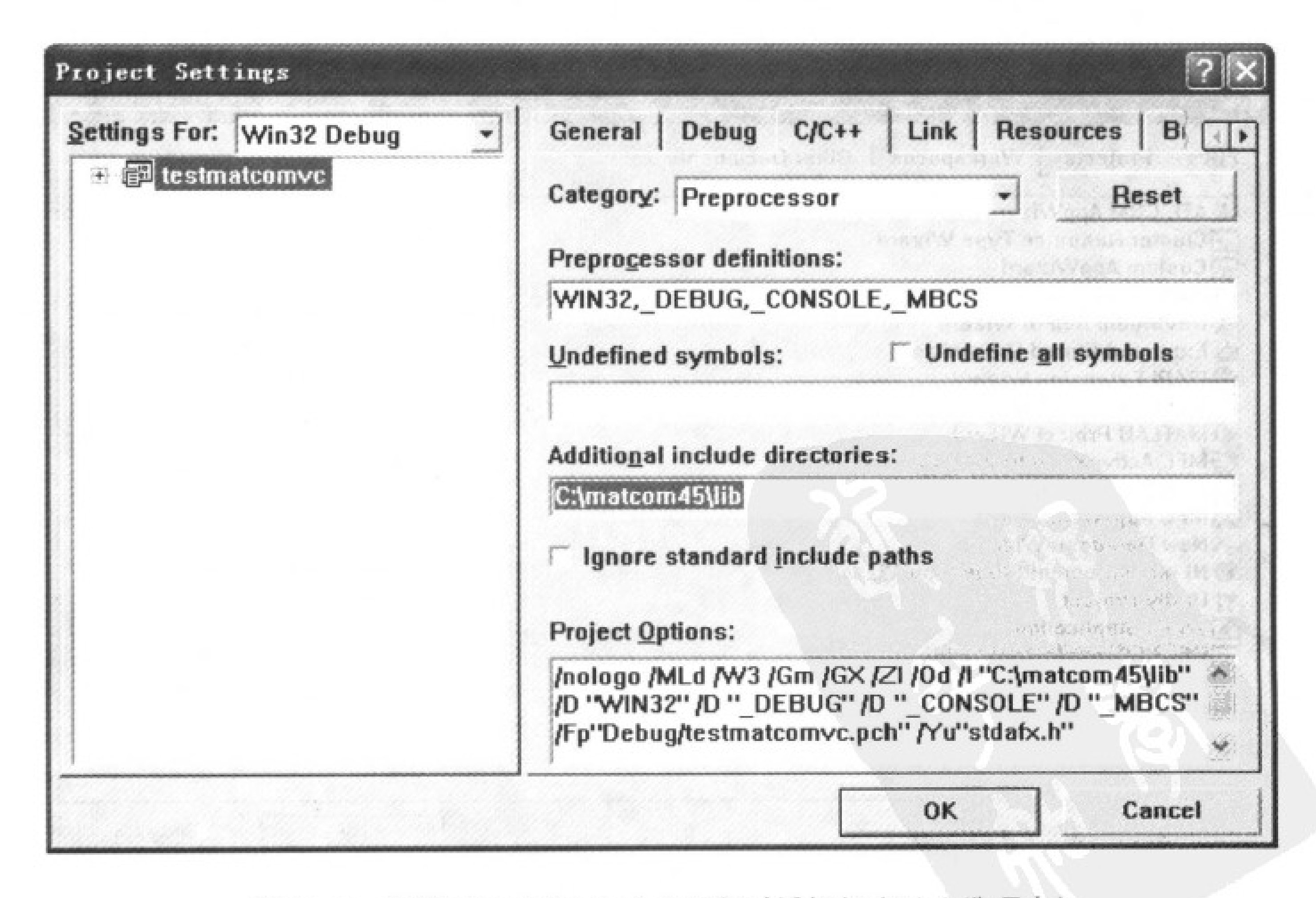


图 7-6 设置 Visual C++ 6.0 工程对话框(C/C++选项卡)

② 加入头文件 matlib. h,运行程序,如果出现错误的话,请仔细查看上述 Visual C++工程设置是否正确。

③ 初始化 Matcom C++矩阵库。Matcom C++矩阵库的初始化十分简单,调用下述语句即可:

```
initM(Matcom_VERSION);
```

Matcom C++矩阵库的初始化和释放是成对进行的,在需要释放 Matcom C++矩阵库时,调用函数 exitM()即可。

Matcom C++矩阵库初始化需要一定的时间,为了加快程序的执行速度,应尽量减少 Matcom C++矩阵库初始化的次数。所以开发人员应仔细考虑 Matcom C++矩阵库生存周期的长度,在 Matcom C++矩阵库生存周期的开始初始化 C++库,在生存周期结束的时候,释放 C++库。这样做虽然会占用一定的内存,但是会提高程序运行的速度。

④ Matcom C++矩阵库矩阵的创建。Matcom C++矩阵库采用面向对象的设计风格,矩阵数据和操作都被封装在类 Mm 中。当需要创建矩阵时,只要声明一个 Mm 对象实例即可。下面是创建矩阵 a、b 和 x 的语句。

```
Mm a;
Mm b;
Mm x;
```

⑤ 初始化矩阵元素和对矩阵元素赋值。在步骤④中创建的矩阵 a、b 和 x 是空的,不包含任何矩阵元素。矩阵在使用以前必须经过初始化,以确定矩阵的维数、各维元素个数及矩阵元素的初始值。初始化矩阵的方式与 Matlab 的语句类似,如:

```
a = rand(3,3);
b = zeros(3,3);
b.r(1,1) = 1;
b.r(2,2) = 3;
b.r(3,3) = 5;
display(a);
display(b);
```

值得注意的是,对于变量 a 的初始化并没有显式地进行,而是等号运算符根据函数 rand 的返回值自动对变量 a 进行初始化。当然,也可以通过显式的方式来实现对变量的初始化,对变量 b 的初始化就是如此。对矩阵元素的赋值是通过索引进行的,其中 r 表示取矩阵的实部。

⑥ 调用 Matcom C++矩阵库的函数。Matcom C++矩阵库提供了一系列矩阵操作的函数 ,其调用方法与一般的 C/C++函数一样。如下所示,为了求解矩阵 a 和 b 的行列式的值,需要调用 det 函数。另外,对于一般矩阵的加、减、乘、除运算,Matcom C++矩阵库重载了所有这些矩阵运算的操作符,以方便使用。

```
x = a + b;
printf("a + b = :\n");
display(x);
```

```
printf("det(a) = :\n");
display(det(a));
printf("det(b) = :\n");
display(det(b));
```

⑦ 得到矩阵的元素。在步骤⑤中使用了“. r(row,col)”对矩阵元素进行赋值,如果要获取矩阵的元素,同样可以使用“. r(row,col)”。例如,可以通过下面的程序片段对一个矩阵 x 的实部进行遍历。需要注意的是,Matcom C++矩阵中,矩阵的索引是从 1 开始计数的。

```
printf("x 矩阵遍历:\n");
for (int i = 1;i <= x.rows();i++)
{
    for (int j = 1;j <= x.cols();j++)
    {
        printf(" %6.2f    ",x.r(i,j));
    }
    printf("\n");
}
```

上述是在 Visual C++工程中使用 Matcom C++矩阵库的步骤,为了方便数据的输出和显示,采用 Win32 Console 的形式。对于其他情况的开发,如 MFC、ActiveX 等,Matcom C++ 矩阵库的使用方法完全相同。将上述语句组合在一起可得如下源代码,请读者按照上述步骤创建一个自己的实例,如果编译并正确输出,读者就已经基本掌握了在 C++中调用 Matcom C++矩阵库的步骤。

```
#include "stdafx.h"
#include "stdio.h"
#include "matlib.h"
int main(int argc, char * argv[])
{
    initM(Matcom_VERSION);
    Mm a;
    Mm b;
    Mm x;
    a = rand(3,3);
    b = zeros(3,3);
    b.r(1,1) = 1;
    b.r(2,2) = 3;
    b.r(3,3) = 5;
    printf("a = :\n");
    display(a);
    printf("b = :\n");
    display(b);
    x = a + b;
```



```
    printf("a+b=:\n");
    display(x);
    printf("det(a)=:\n");
    display(det(a));
    printf("det(b)=:\n");
    display(det(b));
    printf("x 矩阵遍历:\n");
    for (int i=1;i<=x.rows();i++)
    {
        for (int j=1;j<=x.cols();j++)
        {
            printf("%6.2f   ",x.r(i,j));
        }
        printf("\n");
    }
    exitM();
    return 0;
}
```

输出结果如下所示。

```
a=:
ans (3×3)=9 double elements real (72 bytes)=
0.9501   0.486    0.4565
0.2311   0.8913   0.0185
0.6068   0.7621   0.8214
b=:
ans (3×3)=9 double elements real (72 bytes)=
1   0   0
0   3   0
0   0   5
a+b=:
ans (3×3)=9 double elements real (72 bytes)=
1.95     0.486    0.4565
0.2311   3.891    0.0185
0.6068   0.7621   5.821
det(a)=:
ans (1×1)=1 double elements real (8 bytes)=
0.4289
det(b)=:
ans (1×1)=1 double elements real (8 bytes)=
15
x 矩阵遍历:
  1.95     0.49     0.46
```

```
0.23    3.89    0.02
0.61    0.76    5.82
```

7.3 使用 Matcom C++矩阵库的矩阵类 Mm

7.3.1 创建数值矩阵

在 Matcom C++矩阵库中,矩阵作为一个类 Mm 被封装了起来,便于开发人员使用。在 Mm 声明对象实例的时候,构造函数会自动对生成的矩阵对象进行必要的初始化。

矩阵对象的声明和普通的 C++类对象的声明方式相同:

```
Mm a;                       //声明对象 a 为 Mm 矩阵对象
Mm a,b,c;                   //同时声明多个对象
```

默认的矩阵构造函数只是将矩阵对象初始化为一个空的矩阵,例如上例中的对象 a 不含有任何矩阵元素。如果要将矩阵初始化为 m 行,n 列的矩阵,可采用:

```
Mm a;
a = zeros(m,n);
```

此时,a 被初始化为 m 行,n 列的零值矩阵。如果要初始化三维矩阵 m×n×p,可以采用类似的方法,如:

```
Mm a;
a = zeros(m,n,p);
```

如果要创建 m×n 的复数矩阵,其初始化函数则要采用 czeros,如:

```
Mm x;
x = czeros(isc,m,n);        //如果变量 isc 的值是 true 的话,矩阵将被初始化为复数矩阵
```

Mm 类重载了赋值操作符,当某个 Mm 对象被赋值以后,它会被自动初始化并重新分配矩阵的维数和元素的值,因此大部分情况下一个矩阵对象声明以后不需要采用 zeros 或者 czeros 进行初始化。如:

```
Mm a;
a = rand(4,4);              //直接使用并自动初始化为 4×4 的随机数矩阵
```

当然,对于单个数值的情况,可以看做是 1×1 的一个 Mm 矩阵,因而对 Mm 矩阵也可以直接用某个数值进行赋值。如:

```
Mm x1,x2,x3;
x1 = 5;                     //整数直接赋值,自动转换为双精度型
x2 = 3.0;                   //浮点型直接赋值,自动转换为双精度型
x3 = 3 + 4 * i;             //复数直接赋值,实部和虚部直接转换为双精度型
```

Mm 对象可以通过 dMm 宏在构造的时候命名,这样以来当用 display 显示这个变量的时

候，就会将变量的名字放在变量说明的最前面。例如：

```
dMm(noiseMatrix);
noiseMatrix = rands(3,3);
display(noiseMatrix);
```

则输出为(注意加粗的部分，这在以前显示变量的时候是没有的)：

noiseMatrix (3×3) = 9 double elements real (72 bytes) =

```
        0.9501    0.486    0.4565
        0.2311    0.8913   0.0185
        0.6068    0.7621   0.8214
```

变量的名称可以通过.getname()获得，由于 dMm 是一个宏，如果采用这种方式定义变量的话只能采用单独定义的方式，如 dMm(a)、dMm(b)和 dMm(c)。

7.3.2 创建字符矩阵

采用函数 TM 可以创建一个字符矩阵，如下所示：

```
Mm cx;
cx = TM("Hello World! \n");
```

字符矩阵在 Matcom C++矩阵库中是以双精度型保存的，所以字符矩阵和双精度实数矩阵可以相互转换，即矩阵到底是字符型的还是双精度实型的可以动态切换。Matcom C++矩阵库用 setstr 来进行切换。如：

```
Mm a;
a = colon(65,69);          //相当于 Matlab 的 65:69，产生一个向量[65 66 67 68 69];
a.setstr(1);
display(a);                //结果是矩阵中 ASCII 码的字符输出"ABCDE"
```

其中，colon 函数相当于 Matlab 的":"操作符，colon(i,j,k)相当于 i:j:k；colon(i,j)相当于 i:j。以 i:j:k 为例，其含义是产生一个向量，从 i 开始，到 k 结束，步长为 j，如果采用 C 语言来表示，则与下面的语句相等：

```
int m;
for(m = i;m <= k;m = m + j){};
```

将 Matcom C++矩阵库里的字符数组转换为 char * 类型的字符串，采用下面的函数：

```
void Mstr(const Mm& x, char * str, int maxlen);
```

其中，char * str 表示目的字符串，x 表示字符类型的矩阵，maxlen 表示 x 中最大的字符数目，注意字符串最后的"\0"也算一个字符。

7.3.3 利用下标访问矩阵的元素

与 Matlab 中的矩阵元素访问方式类似，可以采用下面几种形式的下标来访问：

```
Mm a;
a=magic(3);                //[8 1 6;3 5 7;4 9 2;];
a.r();                     //8
a.r(2,2);                  //5
a.r(8);                    //7
```

7.3.4 获取矩阵数据的指针

通过函数.addr()和.addi()分别获得矩阵的实部和虚部的地址,其返回值都是 double * 型的指针。需要注意的是,Matcom C++矩阵库与 C/C++数组的组织方式不同,在Matcom C++矩阵库中,数组的组织方式按"列优先"的方式组织,C/C++中按照"行优先"的方式组织,所以复制数据时需要注意。本文后面的实例中给出了一种用 reshape 函数进行"行"优先和"列"优先相互转换的办法。另外,addr 也可以通过下标来得到 Matcom C++矩阵库的矩阵中某个元素的地址,地址类型为 double 型。

例如:

```
Mm x;
double a,b,c;
x=magic(3);                // x=[ 8 1 6 ; 3 5 7 ; 4 9 2 ]
a= *x.addr();              // a=8
b= *x.addr(2);             // b=1
c= *x.addr(2,3);           // c=7
*x.addr(3,1) = -999;
// x=[ 8 1 6 ; 3 5 7 ;-999 9 2 ]
*x.addr() = - *x.addr();
// x=[-8 1 6 ; 3 5 7 ;-999 9 2 ]
```

另外,通过 Matcom C++矩阵库的矩阵指针可以用 memcpy 函数复制矩阵中的数据。

例如:

```
double *
x_data=(double *)malloc(512 * sizeof(double));
Mm x;
x=rand(512,1);
memcpy(x_data,x.addr(),512 * sizeof(double));
```

注意:对于稀疏矩阵,如果要采用索引的方式访问的话,则需要采用 full 函数进行转换。

例如:

```
/* * * * * * * * * * * * * * * * * * * * testmatcomvc.cpp 文件 * * * * * * * * * * * * * * * * */
#include "stdafx.h"
#include "stdio.h"
#include "matlib.h"
int main(int argc, char * argv[])
{
```

```
    initM(MATCOM_VERSION);
    Mm x;
    Mm sparseX;
    x = zeros(4,4);
    x(3) = 1;
    x(7) = 12;
    x(15) = 3;
    display(x);
    sparseX = sparse(x);
    display(sparseX);
    display(full(sparseX(9)));   //索引稀疏矩阵
    exitM();
    return 0;
}
```

7.3.5 Mm 矩阵对象的初始化

Mm 矩阵对象可以通过构造函数的初始化，也可以通过其他具有返回值的函数如函数 rand 来初始化，通过构造函数进行初始化常用的实现方式有：

① 采用 double 型向量直接初始化。

② 采用 double 型常数进行初始化。

③ 采用在 C/C++中构造好的数据块进行初始化。

上述 3 种方式的程序实现如下所示：

```
double data[6] = {5,7.9,5.7,5.0,2.4,-1.6};
Mm x;
M_VECTOR(x,data);                              //形式①，采用 double 型向量直接初始化
Mm y;
y = (BR(2),1,0,semi,3 * i,7,pi,semi,0.6,8,0.1);  //形式②，采用常量
Mm z;
z = zeros(1,6);
memcpy(z.addr(),data,6 * sizeof(double));        //形式③，直接复制数据
```

注意：在形式②中，第一个变量的初始化必须用 BR 函数开头。其中 semi 表示分号(;)，其含义是一行结束，另一行重新开始。在形式③中，需要注意 Matcom Mm 矩阵与 C/C++数组在内存中的存储顺序不一致。在 Matcom Mm 矩阵中，数据按列的方式进行存储；在 C/C++中，数据按照行的方式进行存储。因而需要采用 reshape 函数达到改变 Mm 矩阵数据存储顺序的目的。

7.3.6 Mm 矩阵类的几个常用函数

Mm 矩阵类的所有函数都是以.func(para)的形式调用的，其中常用的函数如下：

- size()　返回矩阵的大小，返回值为 Mm 类型。
- size(dim)　返回矩阵某维的最大元素数。

- rows() 返回矩阵的行数。
- cols() 返回矩阵的列数。
- isstr() 判断矩阵是否是字符矩阵。
- setstr(is) 设置矩阵的字符矩阵特性。
- isc() 判断矩阵是否是复数矩阵。
- getndims() 判断矩阵的维数,返回为 Mm。
- getdims() 判断矩阵的各维大小,返回为 int * 。

下面实例是对上述几个常用函数的具体应用。

```
#include "stdafx.h"
#include "stdio.h"
#include "matlib.h"
int main(int argc, char * argv[])
{
    initM(MATCOM_VERSION);
    Mm x;
    int sizex,rows,cols,ndims, * dims=NULL;
    x=rand(4,6);
    sizex=x.size();
    printf("x 矩阵的元素数为:%d\n",sizex);
    sizex=x.size(2);                        //返回矩阵某维的最大元素数
    printf("x 矩阵的第 2 维的大小为:%d\n",sizex);
    rows=x.rows();                          // 返回矩阵的行数
    printf("矩阵的行数为:%d\n",rows);
    cols=x.cols();                          // 返回矩阵的列数
    printf("矩阵的列数为:%d\n",cols);
    if(x.isstr())                           // 判断矩阵是否是字符矩阵
    {
        printf("x 是字符矩阵! \n");
    }
    else
    {
        printf("x 不是字符矩阵! \n");
    }
    if(x.isc())                             // 判断矩阵是否是复数矩阵
    {
        printf("x 是复数矩阵! \n");
    }
    else
    {
        printf("x 不是复数矩阵! \n");
    }
    ndims=x.getndims();                     // 判断矩阵的维数,返回为 Mm
```

```
    printf("矩阵的维数为：%d\n",ndims);
    dims = x.getdims();                    // 判断矩阵的各维大小,返回为 int *
    for(int i=0;i<ndims;i++)
    {
        printf("第%2d维的大小为：%d\n",i+1,dims[i]);
    }
    exitM();
    return 0;
}
```

输出结果如下所示。

```
x矩阵的元素数为:24
x矩阵的第2维的大小为:6
矩阵的行数为:4
矩阵的列数为:6
x不是字符矩阵!
x不是复数矩阵!
矩阵的维数为:2
第 1维的大小为:4
第 2维的大小为:6
```

7.3.7 Matcom C++矩阵库常量

Matcom C++ 矩阵库中定义了一些常量,其中常用的常量如下：

- i,j　复数使用。
- Pi　圆周率的值。
- Inf　表示无穷大,如果出现 x/0 的情况(x! =0),结果为 Inf。
- NaN　Not-a-Number,0/0 的结果以及其他无意义的算术运算结果。
- eps　最小的数,在 Matcom 中为 2^{-52}。
- semi　表示分号,即矩阵的行结束分隔符。
- c_p　表示冒号操作符,用于 colon 不能用的地方,例如：

```
Mm a,b;
a=(BR(1),4,2,semi,6,3,8);     //a=[1 4 2;6 3 8];
b=a(c_p,2);                   //与 Matlab 语句 b=a(:,2)等价,b 为第二列所有元素[4 3]';
```

- e　自然对数的底 2.718281828459…。
- i_o　当 Matcom C++矩阵库函数需要多个输出时,用于分割输入参数和输出参数,例如：

```
Mm x,u,s,v;
svd(x,i_o,u,s,v);             //其中 x 为输入参数,u、s 和 v 为输出参数
```

具体应用见本章后续关于“多输出”函数使用规范的内容。

7.3.8 调用系统函数

在使用 Matcom C++ 矩阵库时，最常调用的两个函数即初始化和退出矩阵库的函数 initM 和 exitM。这两个函数的调用方式如下：

```
initM(MATCOM_VERSION);
exitM();
```

其中，initM 函数的参数永远是 MATCOM_VERSION，这个宏在 matlib.h 中定义，表明 Matcom C++矩阵库的版本号。另外，需要注意的是 initM 函数可以多次调用，但是无论调用了多少次 initM，exitM 只能调用一次。下面是其他一些系统函数的使用技巧。

1. 如何使 Matcom C++矩阵库同时有多个参数输出？

Matcom C++矩阵库在 C++中不支持返回多个参数的函数调用，为了使函数有多个返回参数，Matcom C++矩阵库采用 i_o 分隔符将输入参数和输出参数分离。如下例所示，假定程序开发人员要调用[v,d]=eig(x)函数，由于 eig 函数同时返回两个输出，所以采用下面的形式调用，其中输入参数 x 和输出参数 v,d 通过分隔符 i_o 隔离，以通知 Matcom C++矩阵库哪些参数是输入参数，哪些参数是输出参数。例如：

```
Mm x,v,d;
x=wilkinson(7);
eig(x,i_o,v,d);
```

2. 如何使函数支持个数不确定的输入参数的情况？

在 Matlab 中通过将输入参数设置为 varargin，可以使函数接受个数不确定的输入参数，而在 Matcom C++矩阵库中，可以借助 CL 函数来实现。例如：

```
Mm x,y;
x=linspace(0,2*pi,100);
y=msin(2*x);
plot((CL(x),y));
plot((CL(x),y,TM("*")));
exitM();
```

3. 如何将矩阵数据存储到 *.mat 文件以及从 *.mat 文件载入？

使用 save 和 load 函数进行矩阵数据的存储和载入，其格式如下：

```
save(filename,(CL(mat0),mat1,…));
load(filename,(CL(mat0),mat1,…));
```

其中，filename 是 Mm 类型的字符矩阵，可以使用 TM 函数直接从 C++字符串中生成。例如：

```
save(TM("mydata.mat"),(CL(x),y));
```

4. 如何将矩阵数据存储为文本文件和从文本文件中读入数据？

使用 fprintf 和 fscanf 函数进行矩阵数据文本格式的操作，其中 fprintf 和 fscanf 的使用方法

与 Matlab 中的函数相似。下面给出分别利用 save 和 fprintf 两个函数进行矩阵数据存储的例子：

```
Mm x,y;
x=linspace(0,2*pi,100);
y=msin(2*x);
//采用 save 函数将矩阵 x,y 的值存储到 temp.mat 文件中(mat 格式)
save(TM("temp.mat"),(CL(x),y));
Mm fid;
fid=fopen(TM("temp.txt"),TM("w"));
//采用 fprintf 函数将矩阵 x,y 的值存储到 temp.txt 文件中(文本格式)
fprintf(fid,TM("%5.2f,%5.2f\n"),(CL(x),y));
fclose(fid);
```

5. 如何进行字符串和矩阵数值的相互转换?

利用 num2str 和 str2num 函数进行字符串和数值的互相转换。例如：

```
Mm x,y;
x=linspace(1,10,10);
y=num2str(x);
char buf[1024];
Mstr(y,buf,1024);
printf("%s\n",buf);
```

输出结果为：

```
1 2 3 4 5 6 7 8 9 10
```

7.4　Matcom C++矩阵库的图形和图像显示功能

利用 Matcom C++矩阵库的图形函数可以实现数据的二维和三维显示，并且可以实现图像的显示。Matcom C++矩阵库的图形函数与 Matlab 的图形函数的名称、属性及使用方法几乎完全一样。正因为如此，Matcom C++矩阵库为图形函数提供的说明比较简单，如果读者在使用过程中碰到问题，可以查看 Matlab 相应的图形函数的使用方法及效果，然后再使用 Matcom 的图形函数即可。

尽管如此，由于 C++语言环境与 Matlab 有很大的不同，因而调用 Matcom C++矩阵库图形函数还是存在如下一些特殊的地方。

1. 输入参数个数不确定

Matlab 图形函数的输入参数的个数大都不确定，因而需要借助 CL 函数来解决函数输入参数的个数不确定的问题。关于 CL 函数的用法，请参照本章“调用系统函数”一节关于 CL 函数用法的讲述。

2. 向 Windows 窗口绘图

Matcom C++矩阵库图形函数可以直接向 Windows 窗口上绘图，因而可以在 Visual

C++ MFC 工程中直接向 MFC 窗口中绘图。其实现方式如下：

```
Mm pos;
//采用 winaxes 函数构造 Matcom 绘图的图形窗口句柄
plothandle = winaxes(m_pMainWnd->m_hWnd);
pos = (BR(100),100,400,200);
//设置 Matcom 窗口的位置
set(plothandle,TM("RealPosition"),pos);
//采用 plot 函数向图形窗口中绘制曲线
plot((CL(rand(1,50))));
```

3. 改变 Matcom 绘图窗口的属性

在使用 Matcom C++矩阵库图形函数时，经常需要用 get 函数得到图形绘制窗口的属性以及用 set 函数设置图形绘制窗口的属性。Matcom C++矩阵库通过图形窗口的属性来控制图形窗口的所有外观显示风格甚至包括图形窗口中显示曲线或者图像的数据。

Matcom C++矩阵库常用的图形窗口有两种，一种是绘制曲线的 Line 类型图形窗口，一种是显示图像的 Image 类型窗口。它们与 Axes 和 Figure 窗口的对应关系如图 7-7 所示，即 Line 窗口和 Image 窗口是从 Axes 窗口派生的，Axes 窗口又是从 Figure 窗口派生的。至于其他的图形窗口如 Patch 等，也都是与 Line 和 Image 窗口处在同一个派生层面上。

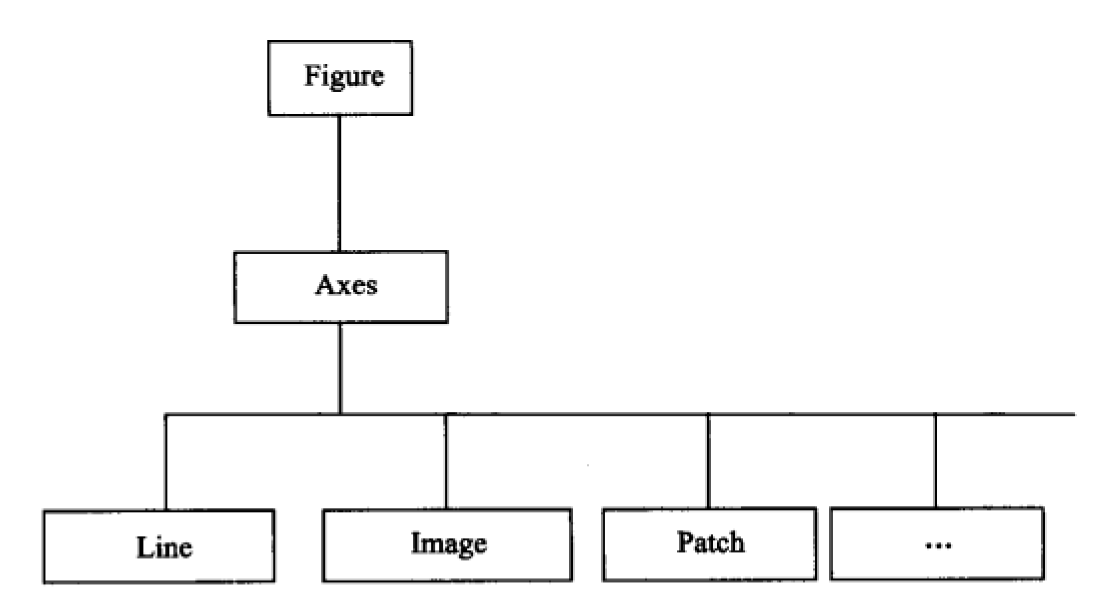


图 7-7　Matcom 主要图形窗口的关系图

Matcom 的 Figure 窗口的句柄被称为 Figure 类型的句柄，Matcom 的 Image 窗口的句柄被称为 Image 类型的句柄，它们都是 Mm 类型的对象。Figure 类型句柄、Axes 类型句柄、Line 类型句柄和 Image 类型句柄的属性有很多，读者可以参考 Matcom 函数说明中关于 Figure，Line 和 Image 等函数的说明，或者参考 Matlab 相关的帮助文档。下面通过几个例子介绍如何得到和改变 Matcom 窗口句柄属性。

```
//gcf 函数可以看做 get current figure 的缩写，其功能是得到当前 Figure 图形窗口的句柄
//隐藏当前 Figure 图形窗口菜单中的 about 项
set(gcf(),TM("MenuAbout"),TM("Off"));
//隐藏当前 Figure 图形窗口菜单中的所有菜单项
set(gcf(),TM("MenuBar"),TM("None"));
```

```
//改变当前窗口的图标，将其更换为 gm.ico 所代表的图标
set(gcf(),TM("IconFile"),TM("gm.ico"));
```

7.5 Matcom 用于图形显示的常用函数

为了便于对照和查阅，表 7-1 中列出了常用的 Matcom 图形显示函数，其中大部分函数的用法与 Matlab 相应同名函数的用法类似，由于 Matcom 函数的说明不是十分详细，因而可以在 Matlab 环境下方便快捷地测试这些函数的主要功能。

表 7-1 常用的 Matcom 图形显示函数

函数名称	函数功能
axis	改变当前坐标轴的最大、最小值
clf,clg	清除当前 Figure 窗口的所有图形对象
close	关闭当前图形窗口
figure	创建一个新的 Figure 窗口，或者切换到指定句柄的 Figure 窗口
grid	在当前绘图窗口中绘制网格
hold	保持当前绘图窗口的内容，使得下次在当前绘图窗口上绘制新图形时，当前已经绘制的内容仍然会保留
print	将当前图形窗口的内容打印到文件或打印机
subplot	在当前 Figure 窗口中创建多个 Axes 绘图窗口对象
title	为当前绘图窗口指定标题
xlabel,ylabel,zlabel	为当前绘图窗口制定 x 轴、y 轴和 z 轴的标签
cla	清除当前 Axes 窗口的所有已经绘制的内容
drawnow	刷新当前图形绘制窗口，使其马上绘制图形窗口数据缓冲区的数据
set	设置指定句柄图形窗口的属性
get	得到指定句柄图形窗口的属性
gca	得到当前 Axes 图形窗口的句柄
gcf	得到当前 Figure 图形窗口的句柄
line	在当前 Axes 窗口中，创建一个 Line 窗口对象
plot	在当前图形窗口中，绘制指定数据的二维曲线
axes	创建一个 Axes 图形窗口，或者切换到指定句柄的 Axes 图形窗口

7.6 Matcom 进行图像显示的常用函数

为了便于对照和查阅，表 7-2 中列出了常用的 Matcom 图像显示函数，这些图像处理函数与其同名的 Matlab 函数的使用方法十分类似。与 Matcom 图形函数类似，读者可以在 Matlab 环境下迅速测试这些函数的主要功能。

表 7-2 常用 Matcom 图像显示函数

函数名称	函数功能
bmpread	读取 bmp 格式的图像
bmpwrite	写入 bmp 格式的图像
getimage	得到图形对象的图像
gifread	读取 gif 格式的图像
gifwrite	写入 gif 格式的图像
im2double	将图像数据从整型转换到 double 型
image	显示位图
imagesc	按比例显示的图像
imfinfo	从图像文件中读取文件信息
imread	读取图像数据
imshow	显示图像数据
imwrite	向文件中写入图像数据
pcxread	读取 pcx 格式的图像数据
pcxwrite	写入 pcx 格式的图像数据
subimage	类似于 subplot,创建当前图形对象的图像显示子对象
tiffread	读取 tiff 格式的图像数据
tiffwrite	写入 tiff 格式的图像数据
truesize	用真实大小显示图形对象中的图像

7.7 Matcom 的应用实例

7.7.1 实例 1 Mm 矩阵的创建及使用

1. 实例说明

本实例主要用来说明 Mm 矩阵类的各种创建及初始化方法以及普通的 Matcom 函数，如 reshape 函数的调用方法。

```
/*****************testmatcomvc.cpp 文件*******************/
#include "stdafx.h"
#include "stdio.h"
#include "matlib.h"
int main(int argc, char *argv[])
{
    initM(Matcom_VERSION);
    //矩阵的索引
    printf("*****Mm 矩阵索引操作*****\n");
```

```
dMm(x);
x=magic(3);
printf("x=magic(3):\n");
display(x);
x.r()=-x.r();
x.r(2)+=1;
x.r(3,1)*=x.r(3,1);
printf("通过索引改变后的 x:\n");
display(x);
//获取矩阵数据的指针
printf("\n*****获取矩阵数据的指针*****\n");
dMm(cdata);
cdata=czeros(1,4,4);
cdata(1)=1+3*i;
cdata(6)=6+4*i;
cdata(11)=2+9*i;
cdata(5)=1+i;
cdata(12)=5+i;
cdata(16)=i;
printf("用 display(data)显示数据:\n");
display(cdata);
double *pr, *pi;
pr=cdata.addr();
pi=cdata.addi();
int i=0,j=0;
printf("获取 data 数据指针后显示:\n");
for(i=0;i<4;i++)
{
    printf("\n");
    for(j=0;j<4;j++)
    {
        if(pi[j*4+i]>0.0000001)
        {
            printf("%1.0f+%1.0fi",pr[j*4+i],pi[j*4+i]);
        }
        else
        {
            printf("%4.0f",pr[j*4+i]);
        }
    }
}
printf("\n");
//直接用 double 型数初始化数组
```

```
    //由于 Mm 矩阵和 C/C++数组的存储顺序不一致,因而需要做相应的转换
    printf("\n*****直接用 double 型数初始化数组*****\n");
    double v_data[10]={1,2,3,4,5,6,7,8,9,10};
    dMm(v);
    M_VECTOR(v,v_data);
    v=reshape(v,5,2);
    v=transpose(v);
    printf("输入的 C 数组:\n");
    for(i=0;i<10;i++)
    {
        printf("%2.0f ",v_data[i]);
    }
    printf("\n 根据输入的 C 数组构造的 Mm 矩阵:\n");
    display(v);
    //通过数据复制直接创建矩阵
    printf("\n*****通过数据复制直接创建矩阵*****\n");
    double v_data1[10]={10,9,8,7,6,5,4,3,2,1};
    memcpy(v.addr(),v_data1,10*sizeof(double));
    v=reshape(v,5,2);
    v=transpose(v);
    printf("输入的 C 数组:\n");
    for(i=0;i<10;i++)
    {
        printf("%2.0f ",v_data1[i]);
    }
    printf("\n 根据输入的 C 数组构造的 Mm 矩阵:\n");
    display(v);
    //矩阵创建的时候直接赋值
    printf("*****矩阵创建的时候直接赋值*****\n");
    dMm (a);
    a=(BR(1),2,3,4,5,semi,6,7,8,9,10,semi);
    display(a);
    printf("采用 colon 函数构造的矩阵索引显示矩阵内容:\n");
    display(colon(1,10));
    exitM();
    return 0;
}
```

2. 实例输出结果

实例程序运行结果如下所示。

```
*****Mm 矩阵索引操作*****
x=magic(3):
```

```
通过索引改变后的 x：
*****获取矩阵数据的指针*****
用 display(data)显示数据：
获取 data 数据指针后显示：
1+3i    1+1i     0       0
 0      6+4i     0       0
 0       0      2+9i     0
 0       0      5+1i    0+1i
*****直接用 double 型数初始化数组*****
输入的 C 数组：
1  2  3  4  5  6  7  8  9 10
根据输入的 C 数组构造的 Mm 矩阵：
*****通过数据复制直接创建矩阵*****
输入的 C 数组：
10  9  8  7  6  5  4  3  2  1
根据输入的 C 数组构造的 Mm 矩阵：
*****矩阵创建的时候直接赋值*****
采用 colon 函数构造的矩阵索引显示矩阵内容：
x (3×3) = 9 double elements real (72 bytes) =
8  1  6
3  5  7
4  9  2
x (3×3) = 9 double elements real (72 bytes) =
-8  1  6
 4  5  7
16  9  2
cdata (4×4) = 16 double elements complex (128 bytes) =
1+3i  1+1i   0     0
 0    6+4i   0     0
 0     0    2+9i   0
 0     0    5+1i  0+1i
v (2×5) = 10 double elements real (80 bytes) =
1   2   3   4   5
6   7   8   9   10
v (2×5) = 10 double elements real (80 bytes) =
10   9   8   7   6
 5   4   3   2   1
a (2×5) = 10 double elements real (80 bytes) =
1   2   3   4   5
6   7   8   9   10
ans (1×10) = 10 double elements real (80 bytes) =
1   2   3   4   5   6   7   8   9   10
```

7.7.2 实例 2 图形绘制的基本功能演示

1. 实例的最终显示界面

实例 2 的运行界面如图 7-8 所示。

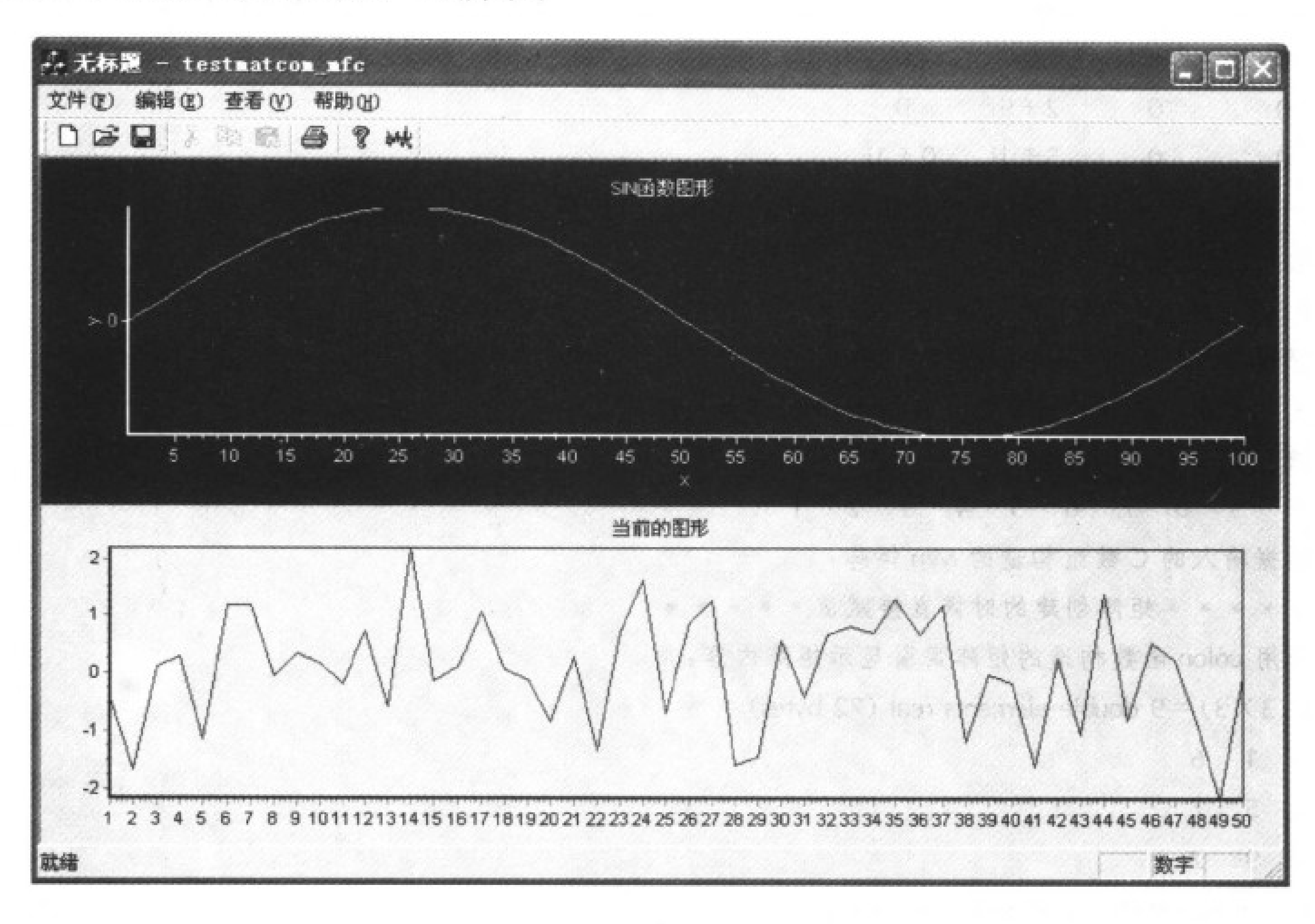


图 7-8 实例 2 的运行界面

2. 实例说明

本实例采用 Matcom 的图形绘制函数在 MFC 的窗口内绘图。

具体操作步骤如下所述。

① 用 Visual C++ 6.0 创建一个单文档的 MFC 工程 testmatcom_mfc。

② 在 View 类中相关定义。

```
Mm m_h;                // 第一个图的 Matcom 句柄
Mm m_data;             //第一图的数据
Mm m_h1;               //第二个图的 Matcom 句柄
Mm m_data1;            //第二个图的数据
BOOL isInitMatcom;     //Matcom 的初始化标志
```

③ 添加 Matcom 及图形绘制函数的初始化函数。

在 view 类的 OnInitialUpdate 函数中加入下列初始化的语句。

```
void CTestmatcom_mfcView::OnInitialUpdate()
{
CView::OnInitialUpdate();
// TODO: Add your specialized code here and/or call the base class
```

```
if(! isInitMatcom)
{
    initM(Matcom_VERSION);
    isInitMatcom=1;
    m_h=winaxes(m_hWnd);
    this->m_hFlag=1;
    axesposition(10,10,100,100);
    double bounddata[4]={0,2*3.1415926,-1,1};
    Mm mbound;
    M_VECTOR(mbound,bounddata);
    //axis
    title((CL(TM("SIN 函数图形"))));
    xlabel((CL(TM("x"))));
    ylabel((CL(TM("y"))));
    set(m_h,(CL(TM("Color")),TM("black")));
    set(m_h,(CL(TM("Box")),TM("on")));
    //axis(mbound);
    Mm x,y;
    x=linspace(0,2*pi,100);
    y=msin(x);
    m_data=y;
    plot((CL(m_data),TM("y")));
    m_h1=winaxes(m_hWnd);
    Mm pos;
    pos=(BR(240),240,200,200);
    set(m_h1,TM("RealPosition"),pos);
    Mm color;
    color=zeros(1,3);
    color.r(1)=0;color.r(2)=0;color.r(3)=0;
    set(m_h,TM("color"),color);
    m_data1=randn(1,50);
    plot((CL(m_data1),TM("b")));
    double *phandle=NULL;
    phandle=m_h1.addr();
    int nrow,ncol;
    nrow=m_h1.rows();
    ncol=m_h1.cols();
    title(CL(TM("当前的图形")));
}
}
```

④ 为了保证绘制图形的大小能够跟随主窗口的变化而变化，在 view 类的 OnSize 函数中加入处理代码。并且 IsMatcomHanleValid 函数将在下面进行说明，其功能是判断当前的

Matcom 图形绘制句柄是否有效。

```
void CTestmatcom_mfcView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    if(this->m_hFlag>0)
    {
        //axes(m_h);
        if(IsMatcomHanleValid(m_h))
        {
            Mm pos;
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=0;
            pos.r(3)=cx;pos.r(4)=cy/2;
            set(m_h,TM("RealPosition"),pos);
        }
        if(IsMatcomHanleValid(m_h1))
        {
            Mm pos;
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=cy/2;
            pos.r(3)=cx;pos.r(4)=cy/2;
            set(m_h1,TM("RealPosition"),pos);
        }
        //axesposition(0,0,cx,cy);
    }
}
```

⑤ 在工具栏上添加一个新的按钮,其 ID 为 ID_DRAWNOISE,然后通过 classwizard 为 view 类添加新的消息响应函数 OnDrawNoise()。这个函数实现的功能为:在第一个图上绘制新的曲线。其实现的代码如下:

```
void CTestmatcom_mfcView::OnDrawnoise()
{
    // TODO: Add your command handler code here
    if(this->isInitMatcom)
    {
        Mm isHold;
        m_data=randn(1,200);
        axes(CL(m_h));
        hold(TM("off"));
        isHold=ishold();
        if((int)(isHold.r(1)))
```

```
            {
            ;
            }
            else
            {
                clf();
                m_h = winaxes(m_hWnd);
                m_h1 = winaxes(m_hWnd);
                CRect rect;
                GetClientRect(&rect);
                int cx = rect.Width();
                int cy = rect.Height();
                if(IsMatcomHanleValid(m_h))
                {
                    Mm pos;
                    pos = zeros(1,4);
                    pos.r(1) = 0;pos.r(2) = 0;
                    pos.r(3) = cx;pos.r(4) = cy/2;
                    set(m_h,TM("RealPosition"),pos);
                }
                axes(CL(m_h));
                plot((CL(m_data),TM("y")));
                title((CL(TM("随机数图形"))));
                xlabel((CL(TM("x"))));
                ylabel((CL(TM("y"))));
                set(m_h,(CL(TM("Color")),TM("black")));
                set(m_h,(CL(TM("Box")),TM("on")));
                if(IsMatcomHanleValid(m_h1))
                {
                    Mm pos;
                    pos = zeros(1,4);
                    pos.r(1) = 0;pos.r(2) = cy/2;
                    pos.r(3) = cx;pos.r(4) = cy/2;
                    set(m_h1,TM("RealPosition"),pos);
                }
                axes(CL(m_h1));
                plot(CL(m_data1));
                //Invalidate(true);
            }
        }
}
```

⑥ 关于 IsMatcomHanleValid 函数的实现。IsMatcomHanleValid 函数用于判断 Matcom

绘图句柄是否有效，由于 Matcom 绘图句柄在失效的时候并不能自动告诉开发者，但是可以通过 Matcom 提供的 isvalid 函数实现 IsMatcomHanleValid 函数的功能。其实现的代码如下。

```
bool CTestmatcom_mfcView::IsMatcomHanleValid(Mm handle)
{
    if(this->isInitMatcom)
    {
        Mm len,isH;
        isH=ishandle(handle);
        len=length(isH);
        if(((int)(len.r(1)))&&((int)(isH.r(1))))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
```

7.7.3 实例 3 利用 Matcom 绘制动态曲线

1. 实例的最终界面

实例 3 运行界面如图 7-9 所示。

2. 实例说明

当读者运行实例 2 时，会发现每次在 Matcom 图形窗口上绘制曲线的时候，窗口内容闪烁得很厉害。本实例利用 set 函数设置 Matcom 图形窗口数据缓冲区数据以及用 drawnow 函数强制 Matcom 图形窗口绘制数据缓冲区的数据可以达到避免图形绘制的时候窗口闪烁的效果。

具体操作步骤如下所述。

① 创建一个 Visual C++ 6.0 单文档工程，为了节省重新配置工程的麻烦，读者可以将上次创建的 testmatcom_mfc 空工程复制一份，然后完成下面的实例。

② 在 view 类中添加如下成员变量。

```
Mm m_h;          //第一个图的 Matcom 句柄
Mm m_data;       //第一图的数据
Mm m_h1;         //第二个图的 Matcom 句柄
Mm m_data1;      //第二个图的数据
```

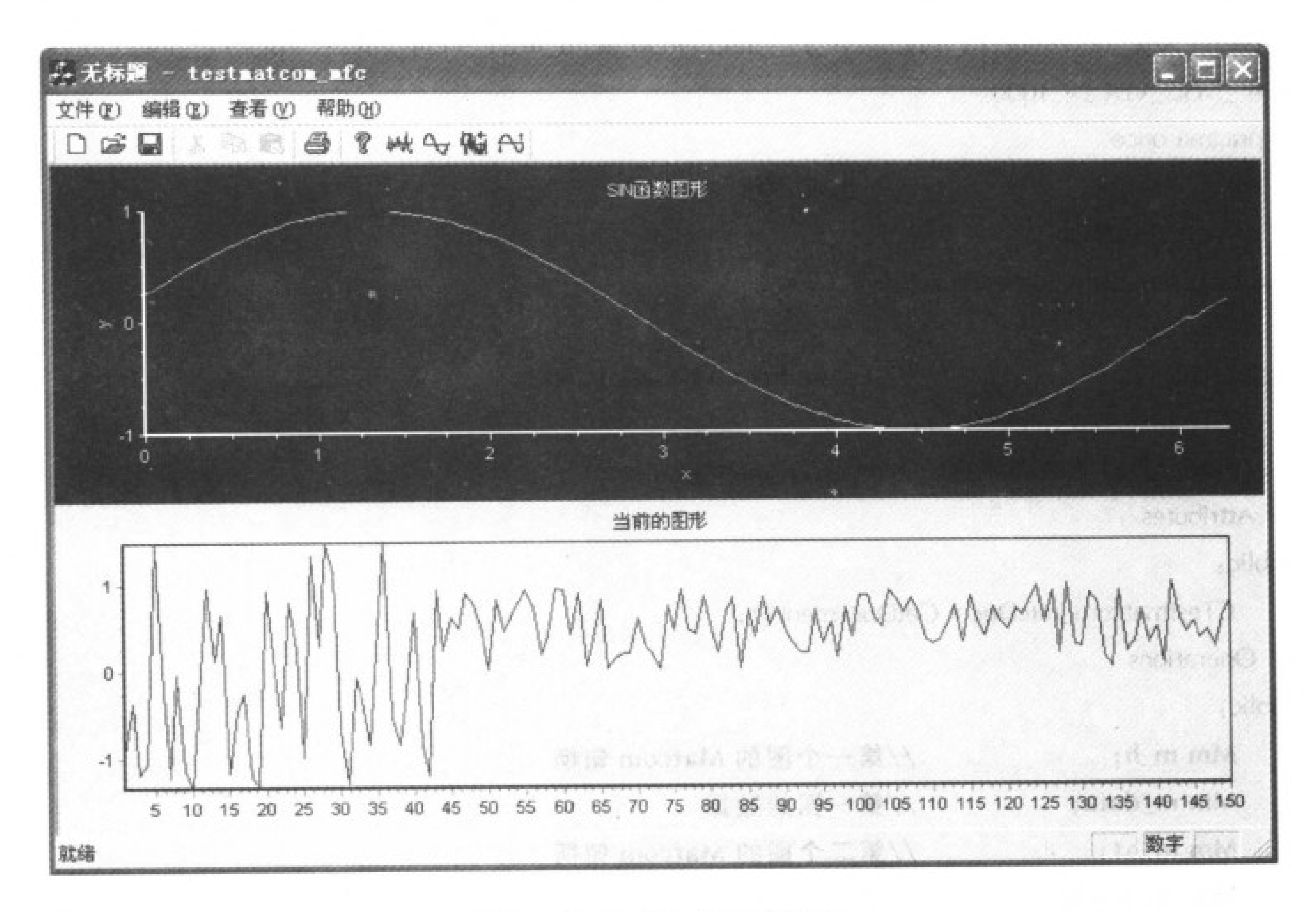


图 7-9 实例 3 的运行界面

```
Mm m_hline1,m_hline;
int m_nScopeTimerID;  //定时器 ID
int m_scopeflag;      //第一个图是否继续显示标志
int m_scopeflag1;     //第二个图是否继续显示标志
int m_hFlag;          //图形窗口是否创建
BOOL isInitMatcom;    //Matcom 的初始化标志
```

③ 在工具栏上添加 4 个按钮，并创建其消息响应函数，按钮的 ID 和消息响应函数的对应关系如表 7-3 所列。

表 7-3 实例 3 的 ID 和消息响应函数的对应关系

ID	消息响应函数
ID_DRAWNOISE	OnDrawnoise
ID_SINGRAPH	OnSingraph
ID_SCOPE1	OnScope1
ID_SCOPE	OnScope

另外，通过 ClassWizard 重载 WM_SIZE 和 WM_TIMER 两个消息的响应函数 OnSize 和 OnTimer。

④ testmatcom_mfcView 类的完整代码如下：

```
/****************** testmatcom_mfcView.h 文件 ****************/
#if ! defined(AFX_TESTMatcom_MFCVIEW_H__52CD9650_BE76_4E81_9154_25C00376E1A1__INCLUDED_)
```

```
#define AFX_TESTMatcom_MFCVIEW_H__52CD9650_BE76_4E81_9154_25C00376E1A1__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif                      // _MSC_VER > 1000
#include "matlib.h"
class CTestmatcom_mfcView : public CView
{
protected:                  // create from serialization only
    CTestmatcom_mfcView();
    DECLARE_DYNCREATE(CTestmatcom_mfcView)
// Attributes
public:
    CTestmatcom_mfcDoc * GetDocument();
// Operations
public:
    Mm m_h;                 //第一个图的 Matcom 句柄
    Mm m_data;              //第一图的数据
    Mm m_h1;                //第二个图的 Matcom 句柄
    Mm m_data1;             //第二个图的数据
    Mm m_hline1,m_hline;
    int m_nScopeTimerID;    //定时器 ID
    int m_scopeflag;        //第一个图是否继续显示标志
    int m_scopeflag1;       //第二个图是否继续显示标志
    int m_hFlag;            //图形窗口是否创建
    BOOL isInitMatcom;      //Matcom 的初始化标志
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestmatcom_mfcView)
    public:
    virtual void OnDraw(CDC * pDC);         // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
    virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
    virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
    //}}AFX_VIRTUAL
// Implementation
public:
    bool IsMatcomHanleValid(Mm handle);
    virtual ~CTestmatcom_mfcView();
#ifdef _DEBUG
    virtual void AssertValid() const;
```

```
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
    //{{AFX_MSG(CTestmatcom_mfcView)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnDrawnoise();
    afx_msg void OnTimer(UINT nIDEvent);
    afx_msg void OnSingraph();
    afx_msg void OnScope1();
    afx_msg void OnScope();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG          // debug version in testmatcom_mfcView.cpp
inline CTestmatcom_mfcDoc * CTestmatcom_mfcView::GetDocument()
   { return (CTestmatcom_mfcDoc * )m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif
// ! defined(AFX_TESTMatcom_MFCVIEW_H__52CD9650_BE76_4E81_9154_25C00376E1A1__INCLUDED_)

/****************testmatcom_mfcView.cpp 文件****************/
#include "stdafx.h"
#include "testmatcom_mfc.h"
#include "testmatcom_mfcDoc.h"
#include "testmatcom_mfcView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView
IMPLEMENT_DYNCREATE(CTestmatcom_mfcView, CView)
BEGIN_MESSAGE_MAP(CTestmatcom_mfcView, CView)
    //{{AFX_MSG_MAP(CTestmatcom_mfcView)
    ON_WM_SIZE()
    ON_COMMAND(ID_DRAWNOISE, OnDrawnoise)
```

```
    ON_WM_TIMER()
    ON_COMMAND(ID_SINGRAPH, OnSingraph)
    ON_COMMAND(ID_SCOPE1, OnScope1)
    ON_COMMAND(ID_SCOPE, OnScope)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView construction/destruction
CTestmatcom_mfcView::CTestmatcom_mfcView()
{
    // TODO: add construction code here
    isInitMatcom = 0;
    m_hFlag = 0;
    m_scopeflag = 0;
    m_nScopeTimerID = 0;
    m_scopeflag1 = 0;
}
CTestmatcom_mfcView::~CTestmatcom_mfcView()
{
    if(isInitMatcom)
    {
        exitM();
        isInitMatcom = 0;
    }
}
BOOL CTestmatcom_mfcView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView drawing
void CTestmatcom_mfcView::OnDraw(CDC * pDC)
{
    CTestmatcom_mfcDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);
```

```
    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView printing
BOOL CTestmatcom_mfcView::OnPreparePrinting(CPrintInfo * pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}
void CTestmatcom_mfcView::OnBeginPrinting(CDC * /* pDC * /, CPrintInfo * / * pInfo * /)
{
    // TODO: add extra initialization before printing
}
void CTestmatcom_mfcView::OnEndPrinting(CDC * / * pDC * /, CPrintInfo * / * pInfo * /)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView diagnostics
#ifdef _DEBUG
void CTestmatcom_mfcView::AssertValid() const
{
    CView::AssertValid();
}
void CTestmatcom_mfcView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
CTestmatcom_mfcDoc * CTestmatcom_mfcView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTestmatcom_mfcDoc)));
    return (CTestmatcom_mfcDoc * )m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_mfcView message handlers
void CTestmatcom_mfcView::OnInitialUpdate()
{
    CView::OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
```

```
if(! isInitMatcom)
{
    initM(Matcom_VERSION);
    isInitMatcom=1;
    m_h=winaxes(m_hWnd);
    this->m_hFlag=1;
    axesposition(10,10,100,100);
    double bounddata[4]={0,2*3.1415926,-1,1};
    Mm mbound;
    M_VECTOR(mbound,bounddata);
    axis(mbound);
    //axis
    title((CL(TM("SIN 函数图形"))));
    xlabel((CL(TM("x"))));
    ylabel((CL(TM("y"))));
    set(m_h,(CL(TM("Color")),TM("black")));
    set(m_h,(CL(TM("Box")),TM("on")));
    //axis(mbound);
    Mm x,y;
    x=linspace(0,2*pi,100);
    y=msin(x);
    m_data=y;
    m_hline=plot((CL(x),m_data,TM("y")));
    m_h1=winaxes(m_hWnd);
    Mm pos;
    pos=(BR(240),240,200,200);
    set(m_h1,TM("RealPosition"),pos);
    Mm color;
    color=zeros(1,3);
    color.r(1)=0;color.r(2)=0;color.r(3)=0;
    set(m_h,TM("color"),color);
    m_data1=randn(1,150);
    m_hline1=plot((CL(m_data1),TM("b")));
    set(m_hline1,TM("Xdata"),linspace(1,150,150));
    set(m_hline1,TM("Ydata"),m_data1);
    drawnow();
    double *phandle=NULL;
    phandle=m_h1.addr();
    int nrow,ncol;
    nrow=m_h1.rows();
    ncol=m_h1.cols();
    title(CL(TM("当前的图形")));
}
```

```
}
void CTestmatcom_mfcView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    //figure(CL(1));
    if(this->m_hFlag>0)
    {
        if(IsMatcomHanleValid(m_h))
        {
            Mm pos;
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=0;
            pos.r(3)=cx;pos.r(4)=cy/2;
            set(m_h,TM("RealPosition"),pos);
        }
        if(IsMatcomHanleValid(m_h1))
        {
            Mm pos;
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=cy/2;
            pos.r(3)=cx;pos.r(4)=cy/2;
            set(m_h1,TM("RealPosition"),pos);
        }
    }
}
void CTestmatcom_mfcView::OnDrawnoise()
{
    // TODO: Add your command handler code here
    if(this->isInitMatcom)
    {
        m_data=randn(1,200);
        if(IsMatcomHanleValid(m_h))
        {
            set(m_hline,TM("Xdata"),linspace(1,200,200));
            set(m_hline,TM("Ydata"),m_data);
            drawnow();
        }
    }
}
bool CTestmatcom_mfcView::IsMatcomHanleValid(Mm handle)
{
    if(this->isInitMatcom)
```

```
    {
        Mm len,isH;
        isH = ishandle(handle);
        len = length(isH);
        if(((int)(len.r(1)))&&((int)(isH.r(1))))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
void CTestmatcom_mfcView::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    if(this->isInitMatcom)
    {
        if(IsMatcomHanleValid(m_h))
        {
            if(this->m_scopeflag1)
            {
                Mm ydata,tmp;
                ydata = get(m_hline1,TM("Ydata"));
                int nydatalen;
                nydatalen = ydata.rows() * ydata.cols();
                for(int i = 1;i<nydatalen;i++)
                {
                    ydata.r(i) = ydata.r(i + 1);
                }
                tmp = randM(1,1);
                ydata.r(nydatalen) = tmp.r(1);
                set(m_hline1,TM("Ydata"),ydata);
                drawnow();
            }
            if(this->m_scopeflag)
            {
                Mm ydata;
```

```
                ydata = get(m_hline,TM("Ydata"));
                int nydatalen;
                double lastYdata;
                nydatalen = ydata.rows() * ydata.cols();
                lastYdata = ydata.r(nydatalen);
                for(int i = nydatalen;i> = 2;i-- )
                {
                    ydata.r(i) = ydata.r(i-1);
                }
                ydata.r(1) = lastYdata;
                set(m_hline,TM("Ydata"),ydata);
                drawnow();
            }
        }
    }
    CView::OnTimer(nIDEvent);
}
void CTestmatcom_mfcView::OnSingraph()
{
    // TODO: Add your command handler code here
    if(this->isInitMatcom)
    {
        if(IsMatcomHanleValid(m_h))
        {
            Mm x,y;
            x = linspace(0,2 * pi,100);
            y = msin(x);
            set(m_hline,TM("Xdata"),x);
            set(m_hline,TM("Ydata"),y);
            drawnow();
        }
    }
}
void CTestmatcom_mfcView::OnScope1()
{
    // TODO: Add your command handler code here
    if(this->isInitMatcom)
    {
        if(IsMatcomHanleValid(m_h1))
        {
            if(m_scopeflag1 == 0)
            {
                Mm ydata,tmp;
```

```
                ydata = get(m_hline1,TM("Ydata"));
                int nydatalen;
                nydatalen = ydata.rows() * ydata.cols();
                for(int i = 1;i<nydatalen;i++)
                {
                    ydata.r(i) = ydata.r(i+1);
                }
                tmp = randM(1,1);
                ydata.r(nydatalen) = tmp.r(1);
                set(m_hline1,TM("Ydata"),ydata);
                drawnow();
                if(m_nScopeTimerID<=0)
                {
                    m_nScopeTimerID = SetTimer(1,50,NULL);
                }
                m_scopeflag1 = 1;
            }
            else
            {
                m_scopeflag1 = 0 ;
            }
        }
    }
}
void CTestmatcom_mfcView::OnScope()
{
    // TODO: Add your command handler code here
    if(this->isInitMatcom)
    {
        if(IsMatcomHanleValid(m_h))
        {
            if(m_scopeflag==0)
            {
                Mm ydata;
                ydata = get(m_hline,TM("Ydata"));
                int nydatalen;
                double lastYdata;
                nydatalen = ydata.rows() * ydata.cols();
                lastYdata = ydata.r(nydatalen);
                for(int i = nydatalen;i>=2;i--)
                {
                    ydata.r(i) = ydata.r(i-1);
                }
```

```
            ydata.r(1) = lastYdata;
            set(m_hline,TM("Ydata"),ydata);
            drawnow();
            if(m_nScopeTimerID<=0)
            {
                m_nScopeTimerID = SetTimer(1,10,NULL);
            }
            m_scopeflag = 1;
        }
        else
        {
            m_scopeflag = 0 ;
        }
    }
  }
}
```

7.7.4　实例 4　利用 Matcom C++矩阵库进行图像显示

1. 实例的最终界面

实例 4 运行结果如图 7－10 所示。

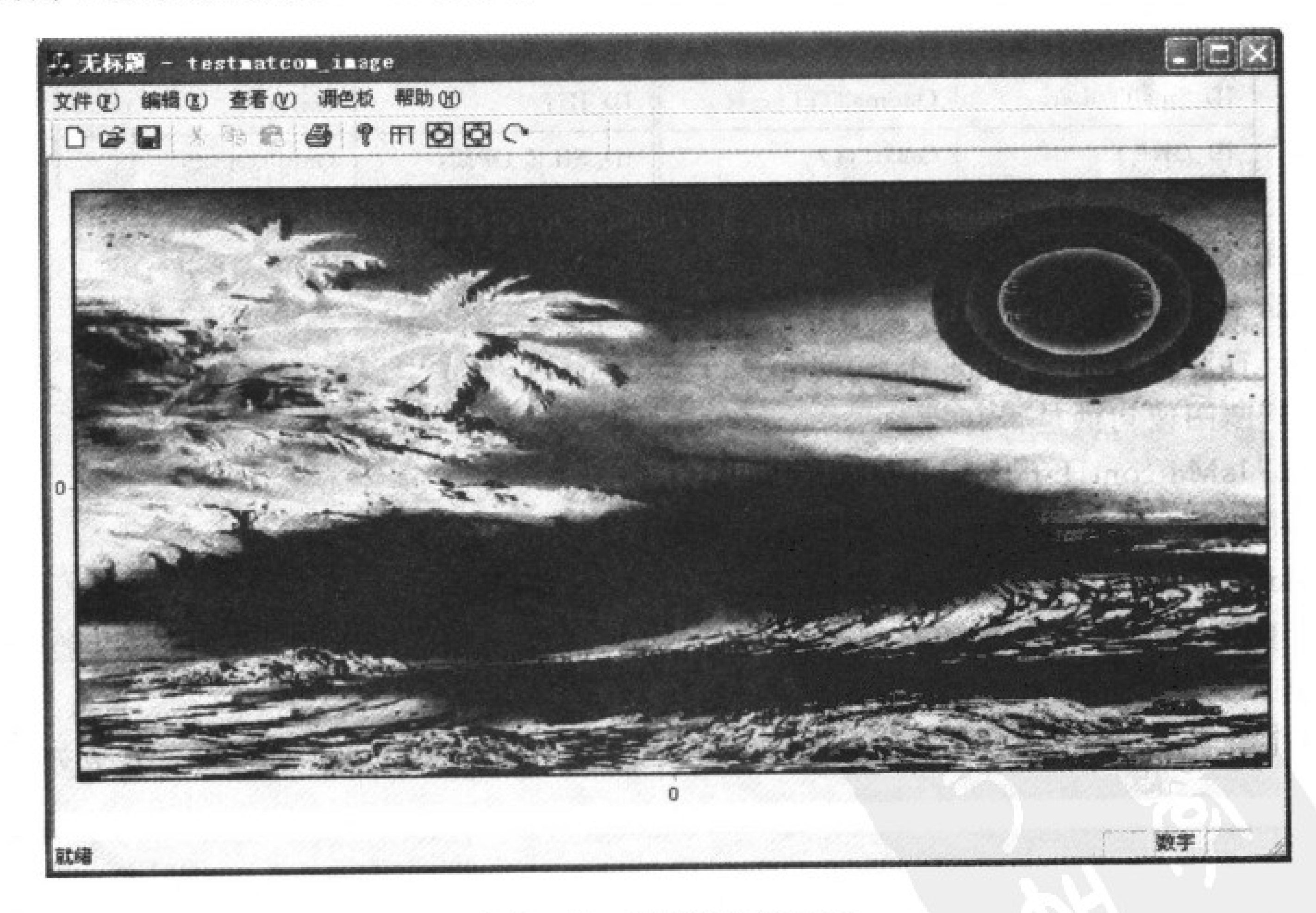


图 7－10　实例 4 运行结果

2. 实例说明

本实例实现的主要功能是：本实例主要实现用 Matcom C++ 矩阵库的图像函数对二维

图像的一般操作。其中包括缩小和放大显示二维图像、改变图像的调色板及采用 Matcom C++矩阵库对图像数据进行运算(FFT)等。

具体操作步骤如下所述。

① 首先创建一个 Visual C++ MFC 单文档工程 testmatcom_image，根据 Matcom C++矩阵库的要求配置好 testmatcom_image 工程。

② 为 testmatcom_imageview 类添加下面的公共成员变量。

```
bool isInit;              //Matcom 初始化标志
Mm m_h;                   //用于显示图像的句柄
Mm m_data, m_map;         //读入图像的数据和调色板数据
//24 位真彩图像到灰度图像的转换函数
void rgb2gray(Mm &m_data,Mm &m_outData);
//判断图像句柄是否有效
bool IsMatcomHandleValid(Mm handle);
//将 CString 类型的字符转换为 Mm 类型的字符串
void ChangeStrToMm(Mm &mstring, CString instring);
```

③ 添加相应的菜单按钮和工具栏按钮，按钮的 ID 及对应的消息响应函数如表 7-4 所列。

表 7-4 testmatcom_image 菜单和工具栏按钮

ID	消息响应函数	ID	消息响应函数
ID_FFT	OnFft()	ID_RESET	OnReset()
ID_LargeToSmall	OnLargeToSmall()	ID_COOL	OnCool()
ID_SmallToLarge	OnSmallToLarge()	ID_JET	OnJet()
ID_GRAY	OnGray()	ID_FILE_OPEN	OnFileOpen()

④ 在 CTestmatcom_imageview 中添加辅助函数。

```
//24 位真彩图像到灰度图像的转换函数
void rgb2gray(Mm &m_data,Mm &m_outData);
//判断图像句柄是否有效
bool IsMatcomHandleValid(Mm handle);
//将 CString 类型的字符转换为 Mm 类型的字符串
void ChangeStrToMm(Mm &mstring, CString instring);
```

⑤ 重载 WM_SIZE 消息响应函数 OnSize()，以实现 Matcom 图形绘制窗口随着 MFC View 窗口大小的变化而变化。

下面给出 testmatcom_imageview.h 和 testmatcom_imageview.cpp 文件的全部代码如下。

```
/****************** testmatcom_imageView.h 文件 ***************/
#if ! defined(AFX_TESTMatcom_IMAGEVIEW_H__C34BC76B_E954_46E8_BAFF_CAF5802EC3BD__INCLUDED_)
#define AFX_TESTMatcom_IMAGEVIEW_H__C34BC76B_E954_46E8_BAFF_CAF5802EC3BD__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif                          // _MSC_VER > 1000
```

```
#include "matlib.h"
class CTestmatcom_imageView : public CView
{
protected:                    // create from serialization only
    CTestmatcom_imageView();
    DECLARE_DYNCREATE(CTestmatcom_imageView)
// Attributes
public:
    CTestmatcom_imageDoc * GetDocument();
// Operations
public:
    bool isInit;              //Matcom 初始化标志
    Mm m_h;                   //用于显示图像的句柄
    Mm m_data, m_map;         //读入图像的数据和调色板数据
    //24 位真彩图像到灰度图像的转换函数
    void rgb2gray(Mm &m_data,Mm &m_outData);
    //判断图像句柄是否有效
    bool IsMatcomHandleValid(Mm handle);
    //将 CString 类型的字符转换为 Mm 类型的字符串
    void ChangeStrToMm(Mm &mstring, CString instring);
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CTestmatcom_imageView)
    public:
    virtual void OnDraw(CDC * pDC);      // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
    virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
    virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
    //}}AFX_VIRTUAL
// Implementation
public:
    virtual ~CTestmatcom_imageView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
    //{{AFX_MSG(CTestmatcom_imageView)
    afx_msg void OnFileOpen();
```

```
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnFft();
    afx_msg void OnLargeToSmall();
    afx_msg void OnSmallToLarge();
    afx_msg void OnReset();
    afx_msg void OnCool();
    afx_msg void OnGray();
    afx_msg void OnJet();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG                    // debug version in testmatcom_imageView.cpp
inline CTestmatcom_imageDoc * CTestmatcom_imageView::GetDocument()
   { return (CTestmatcom_imageDoc * )m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_ LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif
// ! defined(AFX_TESTMatcom_IMAGEVIEW_H__C34BC76B_E954_46E8_BAFF_CAF5802EC3BD__INCLUDED
_)
/ * * * * * * * * * * * * * * testmatcom_imageView.cpp 文件 * * * * * * * * * * * * * * * * * /
#include "stdafx.h"
#include "testmatcom_image.h"
#include "testmatcom_imageDoc.h"
#include "testmatcom_imageView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView
IMPLEMENT_DYNCREATE(CTestmatcom_imageView, CView)
BEGIN_MESSAGE_MAP(CTestmatcom_imageView, CView)
     //{{AFX_MSG_MAP(CTestmatcom_imageView)
     ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
     ON_WM_SIZE()
     ON_COMMAND(ID_FFT, OnFft)
     ON_COMMAND(ID_LargeToSmall, OnLargeToSmall)
     ON_COMMAND(ID_SmallToLarge, OnSmallToLarge)
```

```
    ON_COMMAND(ID_RESET, OnReset)
    ON_COMMAND(ID_COOL, OnCool)
    ON_COMMAND(ID_GRAY, OnGray)
    ON_COMMAND(ID_JET, OnJet)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView construction/destruction
CTestmatcom_imageView::CTestmatcom_imageView()
{
    // TODO: add construction code here
    isInit = false;
    if(isInit == false)
    {
        initM(Matcom_VERSION);
        isInit = true;
    }
}
CTestmatcom_imageView::~CTestmatcom_imageView()
{
    if(isInit)
    {
        exitM();
        isInit = false;
    }
}
BOOL CTestmatcom_imageView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView drawing
void CTestmatcom_imageView::OnDraw(CDC * pDC)
{
```

```
    CTestmatcom_imageDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView printing
BOOL CTestmatcom_imageView::OnPreparePrinting(CPrintInfo * pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}
void CTestmatcom_imageView::OnBeginPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
    // TODO: add extra initialization before printing
}
void CTestmatcom_imageView::OnEndPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView diagnostics
#ifdef _DEBUG
void CTestmatcom_imageView::AssertValid() const
{
    CView::AssertValid();
}
void CTestmatcom_imageView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}
CTestmatcom_imageDoc * CTestmatcom_imageView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTestmatcom_imageDoc)));
    return (CTestmatcom_imageDoc * )m_pDocument;
}
#endif                          //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CTestmatcom_imageView message handlers
//打开图像文件
void CTestmatcom_imageView::OnFileOpen()
```

```
{
    // TODO：Add your command handler code here
    static char BASED_CODE szFilter[]=
        "BMP 格式文件(*.bmp)|*.bmp|所有格式的文件(*.*)|*.*||";
    CFileDialog dlg(1,NULL,NULL,OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,szFilter,this);
    if(dlg.DoModal()==IDOK)
    {
        CString filename=dlg.GetPathName();
        Mm mFileName;
        ChangeStrToMm(mFileName, filename);
        if(! this->IsMatcomHandleValid(m_h))
        {
            m_h=winaxes(this->m_hWnd);
        }
        imread(mFileName,TM("*.bmp"),i_o,m_data,m_map);
        int *pdims=m_data.getdims();
        int ndims=m_data.getndims();
        int nrow=m_data.rows();
        int ncol=m_data.cols();
        m_data=im2double(m_data);
        if(ndims==3 )
        {
            Mm m_data1,maxdata;
            rgb2gray(m_data,m_data1);
            m_data=m_data1;
            maxdata=max(max(max(m_data)));
            m_data=(m_data/maxdata)*255;
        }
        set(m_h,TM("Cdata"),m_data);
        colormap(gray());
        drawnow();
    }
}
//将 C 字符串转换为 Matcom Mm 类型字符串矩阵
void CTestmatcom_imageView::ChangeStrToMm(Mm &mstring, CString instring)
{
    int nlen=instring.GetLength();
    mstring=zeros(1,nlen);
    mstring.setstr(1);
    for(int i=0;i<nlen;i++)
    {
        mstring.r(i+1)=instring.GetAt(i);
    }
```

```
}
void CTestmatcom_imageView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    if(this->IsMatcomHandleValid(this->m_h))
    {
        Mm pos;
        pos = zeros(1,4);
        pos.r(1) = 0;pos.r(2) = 0;
        pos.r(3) = cx;pos.r(4) = cy;
        set(m_h,TM("RealPosition"),pos);
    }
}
//判断 Matcom 图形窗口句柄的有效性
bool CTestmatcom_imageView::IsMatcomHandleValid(Mm handle)
{
    if(this->isInit)
    {
        Mm len,isH;
        isH = ishandle(handle);
        len = length(isH);
        if(((int)(len.r(1)))&&((int)(isH.r(1))))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
//对当前图像进行 FFT,并显示结果
void CTestmatcom_imageView::OnFft()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
```

```
            {
                set(m_h,TM("Cdata"),mabs(fft2(m_data)));
            }
        }
}
//将当前图像缩小显示
void CTestmatcom_imageView::OnLargeToSmall()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            axes(CL(m_h));
            Mm pos;
            Mm xa,ya;
            xa=get(m_h,TM("Xdata"));
            ya=get(m_h,TM("Ydata"));
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=0;
            pos.r(3)=xa.r(2);pos.r(4)=ya.r(2);
            set(m_h,TM("RealPosition"),pos);
        }
    }
}
//将当前图像放大显示
void CTestmatcom_imageView::OnSmallToLarge()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            CRect rect;
            GetClientRect(&rect);
            Mm pos;
            pos=zeros(1,4);
            pos.r(1)=0;pos.r(2)=0;
            pos.r(3)=rect.Width();pos.r(4)=rect.Height();
            set(m_h,TM("RealPosition"),pos);
        }
    }
}
```

```
//显示原始图像
void CTestmatcom_imageView::OnReset()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            set(m_h,TM("Cdata"),m_data);
        }
    }
}
//改变当前调色板为 cool
void CTestmatcom_imageView::OnCool()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            colormap(cool());
        }
    }
}
//将当前输入 RGB24 位真彩图像转换为灰度图像
void CTestmatcom_imageView::rgb2gray(Mm &m_data, Mm &m_outData)
{
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            int ndims;
            ndims = m_data.getndims();
            if(ndims != 3)
            {
                return;
            }
            else
            {
                m_outData = zeros(m_data.rows(),m_data.cols());
                for(int i = 1;i<= m_data.rows();i++)
                {
                    for(int j = 1;j<= m_data.cols();j++)
```

```
                {
                    m_outData.r(i,j) = m_data.r(i,j,1) * 0.3 + m_data.r(i,j,2) * 0.587 + m_
                            data.r(i,j,3) * 0.114;
                }
            }
        }
    }
}
}
//将当前图像调色板转换为灰度调色板
void CTestmatcom_imageView::OnGray()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            colormap(gray());
        }
    }
}
//将当前调色板转换为 jet 调色板
void CTestmatcom_imageView::OnJet()
{
    // TODO: Add your command handler code here
    if(this->isInit)
    {
        if(this->IsMatcomHandleValid(this->m_h))
        {
            colormap(jet());
        }
    }
}
```

7.7.5　实例 5　Matcom 二维和三维曲线绘制综合应用

1. 实例最后结果

实例 5 运行结果如图 7－11 所示。

2. 实例说明

本实例用来说明采用 Matcom C++ 矩阵库的图形函数进行各种类型的数据显示功能，其中包括极坐标、三维数据显示、三维数据等高线显示、二维数据显示、二维数据条形图及阴影显示等。

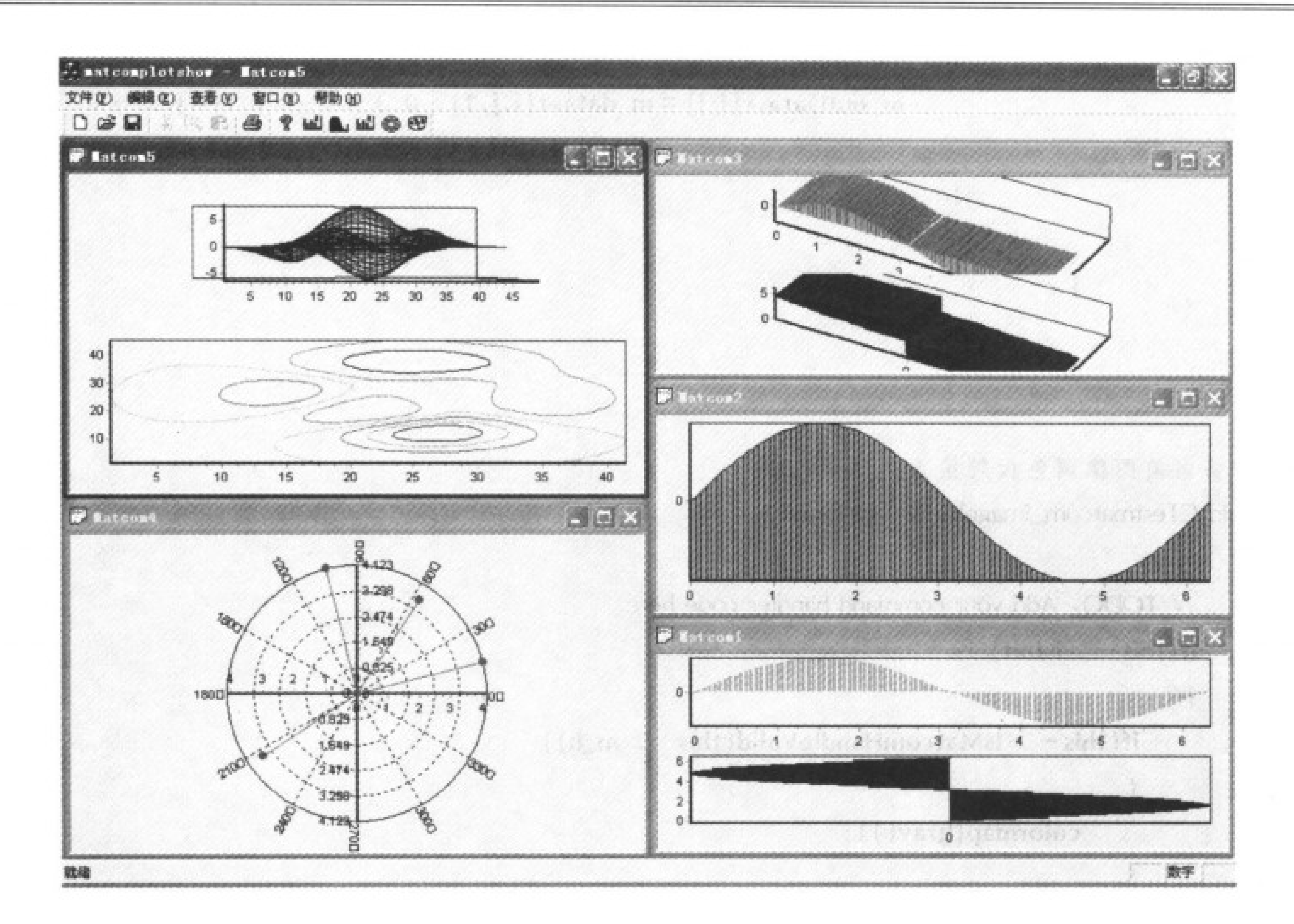


图 7-11　实例 5 运行结果

具体操作步骤如下所述。

① 建立一个 Visual C++ MFC 多文档工程 matcomplotshow。

② 添加相应的工具条按钮及其消息响应函数,如表 7-5 所列。

表 7-5　实例 matcomplotshow 工具条按钮及对应的消息响应函数

ID	消息响应函数
ID_2DBAR	On2dbar()
ID_2DAREA	On2darea()
ID_3DBAR	On3dbar()
ID_2DCOMPASS	On2dcompass
ID_3DCONTOUR	On3dcontour

③ 重载 WM_SIZE 的消息响应函数 OnSize 及 OnInitialUpdate 函数。

④ 添加下列辅助函数:

```
void ClearData(Mm m_h);                //清除句柄为 m_h 的图形窗口的数据
void ResizePlot();                     //重新布置当前 Matcom 的图形窗口
bool IsMatcomHandleValid(Mm m_h);      //判断 Matcom 图形句柄是否有效
```

matcomplotshowview.h 及 matcomplotshowview.cpp 的所有源代码的实现如下所示。

```
/****************** matcomplotshowView.h 文件 ******************/
#if ! defined(AFX_MatcomPLOTSHOWVIEW_H__61083116_F16E_4166_A7E0_7135FDF4E448__INCLUDED_)
```

```
#define AFX_MatcomPLOTSHOWVIEW_H__61083116_F16E_4166_A7E0_7135FDF4E448__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif                          // _MSC_VER > 1000
#include "matlib.h"
class CMatcomplotshowView : public CView
{
protected:                      // create from serialization only
    CMatcomplotshowView();
    DECLARE_DYNCREATE(CMatcomplotshowView)
// Attributes
public:
    CMatcomplotshowDoc * GetDocument();
// Operations
public:
    Mm m_h,m_hplot;             //第一个 Matcom 图形窗口的 Figure 句柄和 Axes 句柄
    Mm m_h1,m_hplot1;           //第二个 Matcom 图形窗口的 Figure 句柄和 Axes 句柄
    bool isTwoPlot;             //当前 View 中是否有两个 Matcom 图形绘制窗口
    bool isInit;                //Matcom C++矩阵库是否初始化
    Mm m_2ddata;                //二维图形 y 轴数据
    Mm m_2ddatax;               //二维图形 x 轴数据
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMatcomplotshowView)
    public:
    virtual void OnDraw(CDC * pDC);   // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void OnInitialUpdate();
    protected:
    virtual BOOL OnPreparePrinting(CPrintInfo * pInfo);
    virtual void OnBeginPrinting(CDC * pDC, CPrintInfo * pInfo);
    virtual void OnEndPrinting(CDC * pDC, CPrintInfo * pInfo);
    //}}AFX_VIRTUAL
// Implementation
public:
    void ClearData(Mm m_h);
    void ResizePlot();
    bool IsMatcomHandleValid(Mm m_h);
    virtual ~CMatcomplotshowView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
```

```
protected:
// Generated message map functions
protected:
    //{{AFX_MSG(CMatcomplotshowView)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void On2dbar();
    afx_msg void On3dbar();
    afx_msg void On2darea();
    afx_msg void On2dcompass();
    afx_msg void On3dcontour();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG                                // debug version in matcomplotshowView.cpp
inline CMatcomplotshowDoc * CMatcomplotshowView::GetDocument()
   { return (CMatcomplotshowDoc * )m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif
// ! defined(AFX_MatcomPLOTSHOWVIEW_H__61083116_F16E_4166_A7E0_7135FDF4E448__INCLUDED_)

/*************** matcomplotshowView.cpp 文件 ***************/
#include "stdafx.h"
#include "matcomplotshow.h"
#include "matcomplotshowDoc.h"
#include "matcomplotshowView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMatcomplotshowView
IMPLEMENT_DYNCREATE(CMatcomplotshowView, CView)
BEGIN_MESSAGE_MAP(CMatcomplotshowView, CView)
    //{{AFX_MSG_MAP(CMatcomplotshowView)
    ON_WM_SIZE()
    ON_COMMAND(ID_2DBAR, On2dbar)
    ON_COMMAND(ID_3DBAR, On3dbar)
```

```
    ON_COMMAND(ID_2DAREA, On2darea)
    ON_COMMAND(ID_2DCOMPASS, On2dcompass)
    ON_COMMAND(ID_3DCONTOUR, On3dcontour)
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMatcomplotshowView construction/destruction
CMatcomplotshowView::CMatcomplotshowView()
{
    // TODO: add construction code here
    isInit = false;
    isTwoPlot = false;
    if(! isInit)
    {
        initM(Matcom_VERSION);
        isInit = true;
    }
}
CMatcomplotshowView::~CMatcomplotshowView()
{
}
BOOL CMatcomplotshowView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMatcomplotshowView drawing
void CMatcomplotshowView::OnDraw(CDC * pDC)
{
    CMatcomplotshowDoc * pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
```

```
// CMatcomplotshowView printing
BOOL CMatcomplotshowView::OnPreparePrinting(CPrintInfo * pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}
void CMatcomplotshowView::OnBeginPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
    // TODO: add extra initialization before printing
}
void CMatcomplotshowView::OnEndPrinting(CDC * /* pDC */, CPrintInfo * /* pInfo */)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CMatcomplotshowView diagnostics
#ifdef _DEBUG
void CMatcomplotshowView::AssertValid() const
{
    CView::AssertValid();
}
void CMatcomplotshowView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CMatcomplotshowDoc * CMatcomplotshowView::GetDocument()      // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CMatcomplotshowDoc)));
    return (CMatcomplotshowDoc *)m_pDocument;
}
#endif                              //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMatcomplotshowView message handlers
bool CMatcomplotshowView::IsMatcomHandleValid(Mm handle)
{
    if(this->isInit)
    {
        Mm len,isH;
        isH = ishandle(handle);
        len = length(isH);
```

```
        if(((int)(len.r(1)))&&((int)(isH.r(1))))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
void CMatcomplotshowView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    // TODO: Add your message handler code here
    if(isInit)
    {
        if(!isTwoPlot)
        {
            if(this->IsMatcomHandleValid(m_h))
            {
                Mm position=zeros(1,4);
                position.r(1)=0;
                position.r(2)=0;
                position.r(3)=cx;
                position.r(4)=cy;
                set(m_h,TM("RealPosition"),position);
            }
            if(this->IsMatcomHandleValid(m_h1))
            {
                Mm position=zeros(1,4);
                position.r(1)=0;
                position.r(2)=0;
                position.r(3)=0;
                position.r(4)=0;
                set(m_h1,TM("RealPosition"),position);
            }
        }
        else
        {
```

```
            if(this->IsMatcomHandleValid(m_h))
            {
                Mm position=zeros(1,4);
                position.r(1)=0;
                position.r(2)=0;
                position.r(3)=cx;
                position.r(4)=cy/2;
                set(m_h,TM("RealPosition"),position);
            }
            if(this->IsMatcomHandleValid(m_h1))
            {
                Mm position=zeros(1,4);
                position.r(1)=0;
                position.r(2)=cy/2;
                position.r(3)=cx;
                position.r(4)=cy/2;
                set(m_h1,TM("RealPosition"),position);
            }
        }
    }
}
void CMatcomplotshowView::OnInitialUpdate()
{
    CView::OnInitialUpdate();
    // TODO: Add your specialized code here and/or call the base class
    m_h1=winaxes(this->m_hWnd);
    Mm position=zeros(1,4);
    position.r(1)=0;
    position.r(2)=0;
    position.r(3)=0;
    position.r(4)=0;
    set(m_h1,TM("RealPosition"),position);
    m_h=winaxes(this->m_hWnd);
    m_2ddatax=linspace(0,2*pi,100);
    m_2ddata=msin(m_2ddatax);
    m_hplot=plot((CL(m_2ddatax),m_2ddata));
    axes(CL(m_h1));
    m_hplot1=plot((CL(m_2ddatax),m_2ddata));
    set(m_h,TM("NextPlot"),TM("replace"));
    set(m_h1,TM("NextPlot"),TM("replace"));
}
//测试二维数据条形图显示
void CMatcomplotshowView::On2dbar()
```

```
{
    // TODO: Add your command handler code here
    if(isInit)
    {
        if(this->IsMatcomHandleValid(m_h))
        {
            axes(CL(m_h));
            Mm color=zeros(1,3);
            color.r(1)=0;
            color.r(2)=1;
            color.r(2)=1;
            ClearData(m_hplot);
            m_hplot=bar(m_2ddatax,m_2ddata);
            set(m_hplot,TM("color"),color);
            axes(CL(m_h1));
            ClearData(m_hplot1);
            m_2ddatax=linspace(0,2*pi,50);
            m_2ddata=msin(m_2ddatax);
            m_hplot1=barh(m_2ddatax,m_2ddata,TM("b"));
            this->isTwoPlot=true;
            ResizePlot();
        }
    }
}
//测试三维数据条形图显示
void CMatcomplotshowView::On3dbar()
{
    // TODO: Add your command handler code here
    if(isInit)
    {
        if(this->IsMatcomHandleValid(m_h))
        {
            axes(CL(m_h));
            Mm color=zeros(1,3);
            color.r(1)=0;
            color.r(2)=1;
            color.r(2)=1;
            ClearData(m_hplot);
            m_hplot=bar3(m_2ddatax,m_2ddata);
            set(m_hplot,TM("color"),color);
            axes(CL(m_h1));
            ClearData(m_hplot1);
            set(m_hplot1,TM("color"),color);
```

```
                m_2ddatax = linspace(0,2 * pi,50);
                m_2ddata = msin(m_2ddatax);
                m_hplot1 = bar3h(m_2ddatax,m_2ddata,TM("b"));
                this->isTwoPlot = true;
                ResizePlot();
            }
        }
}
//重新布置当前 Matcom 的图形窗口
void CMatcomplotshowView::ResizePlot()
{
    if(isInit)
    {
        CRect rect;
        GetClientRect(&rect);
        int cx = rect.Width();
        int cy = rect.Height();
        if(! isTwoPlot)
        {
            if(this->IsMatcomHandleValid(m_h))
            {
                Mm position = zeros(1,4);
                position.r(1) = 0;
                position.r(2) = 0;
                position.r(3) = cx;
                position.r(4) = cy;
                set(m_h,TM("RealPosition"),position);
            }
            if(this->IsMatcomHandleValid(m_h1))
            {
                Mm position = zeros(1,4);
                position.r(1) = 0;
                position.r(2) = 0;
                position.r(3) = 0;
                position.r(4) = 0;
                set(m_h1,TM("RealPosition"),position);
            }
        }
        else
        {
            if(this->IsMatcomHandleValid(m_h))
            {
                Mm position = zeros(1,4);
```

```
                position.r(1)=0;
                position.r(2)=0;
                position.r(3)=cx;
                position.r(4)=cy/2;
                set(m_h,TM("RealPosition"),position);
            }
            if(this->IsMatcomHandleValid(m_h1))
            {
                Mm position=zeros(1,4);
                position.r(1)=0;
                position.r(2)=cy/2;
                position.r(3)=cx;
                position.r(4)=cy/2;
                set(m_h1,TM("RealPosition"),position);
            }
        }
    }
}
//清空当前图形窗口缓冲区的数据
void CMatcomplotshowView::ClearData(Mm handle)
{
    if(isInit)
    {
        if(this->IsMatcomHandleValid(handle))
        {
            Mm temp;
            set(handle,TM("Xdata"),temp);
            set(handle,TM("Ydata"),temp);
            set(handle,TM("Zdata"),temp);
        }
    }
}
//二维数据阴影显示
void CMatcomplotshowView::On2darea()
{
    // TODO: Add your command handler code here
    if(isInit)
    {
        if(this->IsMatcomHandleValid(m_h))
        {
            axes(CL(m_h));
            Mm color=zeros(1,3);
            color.r(1)=0;
```

```
                color.r(2) = 1;
                color.r(2) = 1;
                ClearData(m_hplot);
                m_hplot = area((CL(m_2ddatax),m_2ddata));
                set(m_hplot,TM("color"),color);
                this->isTwoPlot = false;
                ResizePlot();
            }
        }
}
//二维数据极坐标显示
void CMatcomplotshowView::On2dcompass()
{
    // TODO: Add your command handler code here
    if(isInit)
    {
        if(this->IsMatcomHandleValid(m_h))
        {
            axes(CL(m_h));
            Mm color = zeros(1,3);
            color.r(1) = 1;
            color.r(2) = 0;
            color.r(3) = 0;
            ClearData(m_hplot);
            Mm xdata,ydata;
            xdata = zeros(1,4);
            ydata = zeros(1,4);
            xdata.r(1) = 1;
            xdata.r(2) = -2;
            xdata.r(3) = 3;
            xdata.r(4) = 4;
            ydata.r(1) = 4;
            ydata.r(2) = -3;
            ydata.r(3) = 2;
            ydata.r(4) = -1;
            m_hplot = compass(xdata,ydata);
            set(m_hplot,TM("color"),color);
            this->isTwoPlot = false;
            ResizePlot();
        }
    }
}
//三维数据网格及等高线显示
```

```
void CMatcomplotshowView::On3dcontour()
{
    // TODO: Add your command handler code here
    if(isInit)
    {
        if(this->IsMatcomHandleValid(m_h))
        {
            axes(CL(m_h));
            ClearData(m_hplot);
            Mm data=peaks();
            m_hplot=mesh(CL(data));
            view(10, 0);
            this->isTwoPlot=true;
            Mm color;
            color=zeros(1,3);
            color.r(1)=0;
            color.r(2)=0;
            color.r(3)=1;
            set(m_hplot,TM("color"),color);
            axes(CL(m_h1));
            ClearData(m_hplot1);
            m_hplot1=contour(CL(data));
            ResizePlot();
        }
    }
}
```



第8章　Visual C++调用 Matlab C++数学库

8.1　Matlab C++ 数学库介绍

Matlab C++数学库包含了四百多个常用 Matlab 数学函数，并且其调用方式和 Matlab 函数的使用习惯极其相似。对于 Matlab 的使用者来说，采用 Matlab C++数学库，可以使应用程序完全脱离 Matlab 的解释环境。对于 C++的使用者来说，采用 Matlab C++数学库可以充分利用 Matlab 已有的矩阵运算的数学函数库，加快程序的开发进度。

对带有 Matlab C++数学库的 Matlab 6.5 版本，Matlab 数学库安装以后，在<Matlab 根目录>\extern\include\cpp 目录下可以看到 matlab.hpp 和 libmwsglm.hpp 两个文件。

8.2　在 Visual C++工程中调用 Matlab C++数学库

如果要在 Visual C++工程中调用 Matlab C++数学库，需要按照下面 3 个步骤改变 Visual C++工程创建时的默认设置。

1. 需要加入和忽略的静态链接库

对于使用 Matlab C++ 数学库的 Visual C++工程人员来说，在 Visual C++工程设置中需要加入的静态链接库有：libmatpm.lib、libmx.lib、libmatlb.lib、libmat.lib、libmmfile.lib、sgl.lib 和 libmwsglm.lib，如图 8-1 所示。其中 sgl.lib 和 libmwsglm.lib 只有在用到 Matlab C++图形库时才需要在 VC++工程设置中加入。

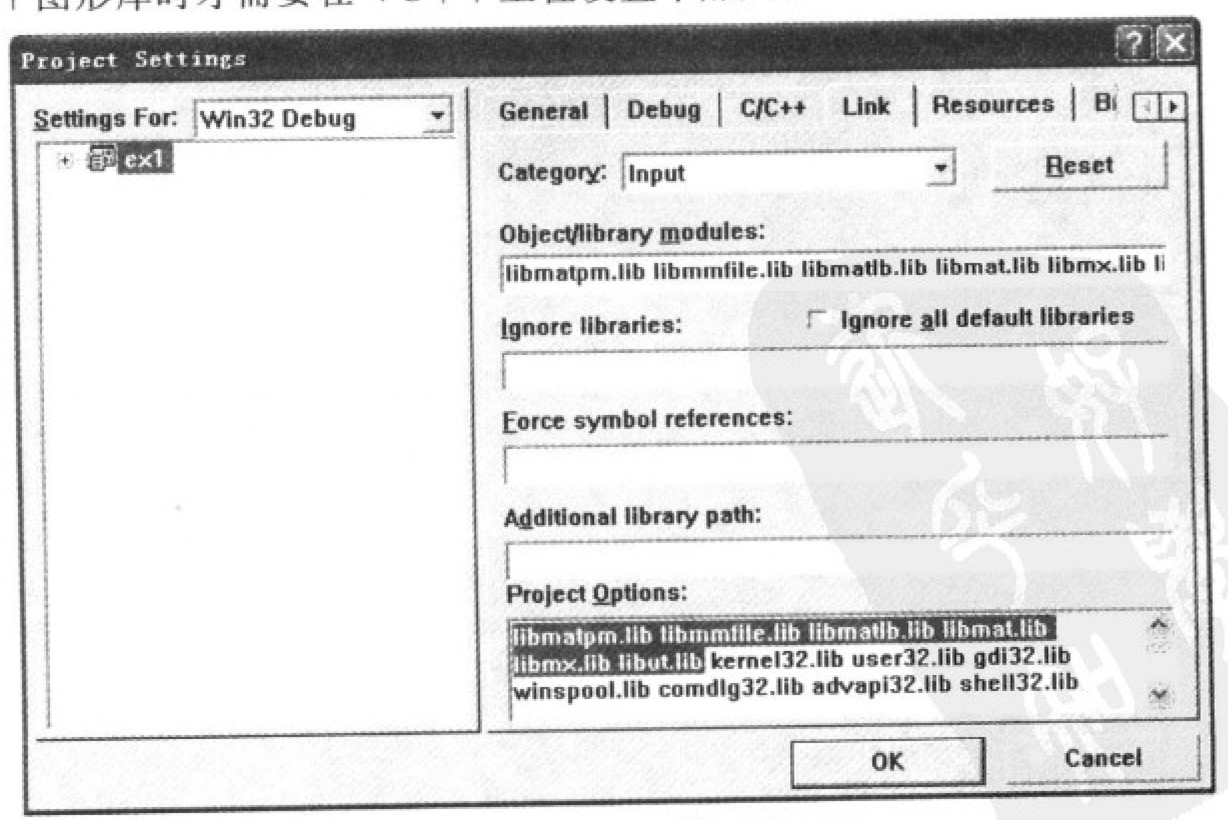


图 8-1　需要加入的静态链接库

另外，在 Visual C++工程设置中需要忽略 VC++的 MSVCRT 静态链接库。

2. 设置 C/C++选项卡中的选项

在 Category 的下拉列表框中选择 Code Generation，在 Use run-time library 下拉列表框中选择 Multithreaded DLL。如图 8－2 所示。

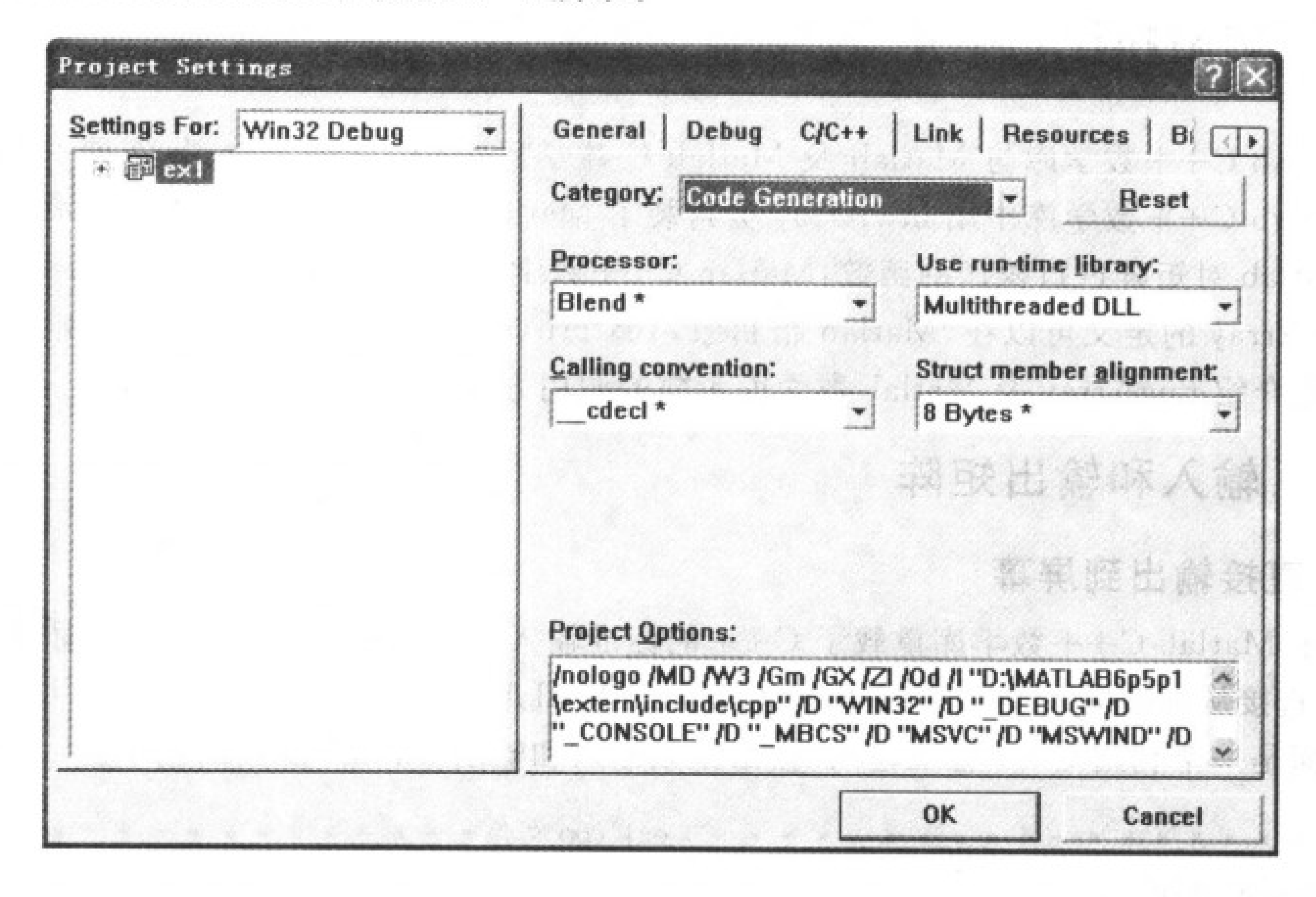


图 8－2　改变 Run-time library 的设置

在 Category 的下拉列表框中选择 Peprocessor，并且在 Preprocessor definitions 文本框中增加如下内容：MSVC，MSWIND，IBMPC，D__STDC_，_AFXDLL。如图 8－3 所示。

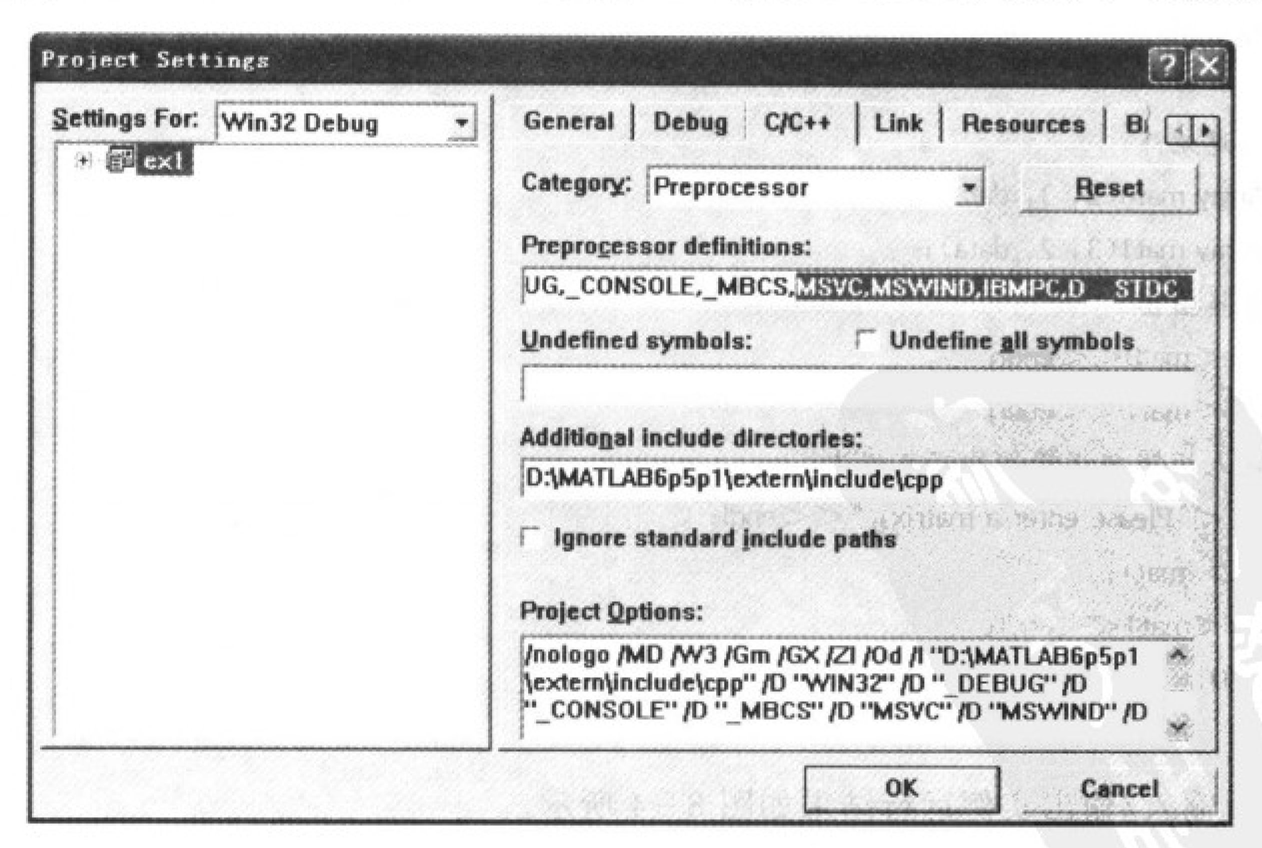


图 8－3　增加预定义宏

3. 在工程中包含相应的头文件

在 Visual C++工程中如果不用 Matlab C++图形库,则只需要包含头文件matlab. hpp;如果要用到 Matlab C++图形库,则需要包含头文件 matlab. hpp 和 libmwsglm. hpp。

8.3 Matlab C++数学库的使用

Matlab C++数学库与 Matlab 及 Matlab C 数学库类似,只有 Matlab 阵列一种数据类型。Matlab C++数学库中用 mwArray 类封装了 Matlab 阵列的数据及其基本操作函数。而对于 Matlab 对矩阵进行操作的函数,Matlab C++数学库通过提供外部函数的方式供用户调用。mwArray 的定义可以在〈Matlab 根目录〉\extern\include\cpp\ dblmtrx. h 文件中找到。下面分别介绍 mwArray 及 Matlab 数学库函数的调用方法。

8.3.1 输入和输出矩阵

1. 直接输出到屏幕

由于 Matlab C++数学库重载了 C++的标准输入 cin 和输出函数 cout,因而可以用 cin 和 cout 直接从标准输入/输出设备中输入和输出 Matlab 阵列。

下面是一个利用 C++标准输入/输出函数输入和输出 Matlab 阵列的例子。

```
/* * * * * * * * * * * * * * * * * * * * * * * ex1.cpp 文件 * * * * * * * * * * * * * * * * * * */
#include "stdafx.h"
#include "matlab.hpp"
#include <stdlib.h>
static double data[] = { 1, 2, 3, 4, 5, 6 };
int main(void)
{
  // 创建矩阵
  mwArray mat0(2, 3, data);
  mwArray mat1(3, 2, data);
  // 矩阵输出
  cout<<mat0<<endl;
  cout<<mat1<<endl;
  // 输入矩阵并将输入的矩阵输出
  cout<<"Please enter a matrix: "<<endl;
  cin >> mat1;
  cout<<mat1<<endl;
  return 0;
}
```

Matlab 输入/输出实例运行结果如图 8-4 所示。

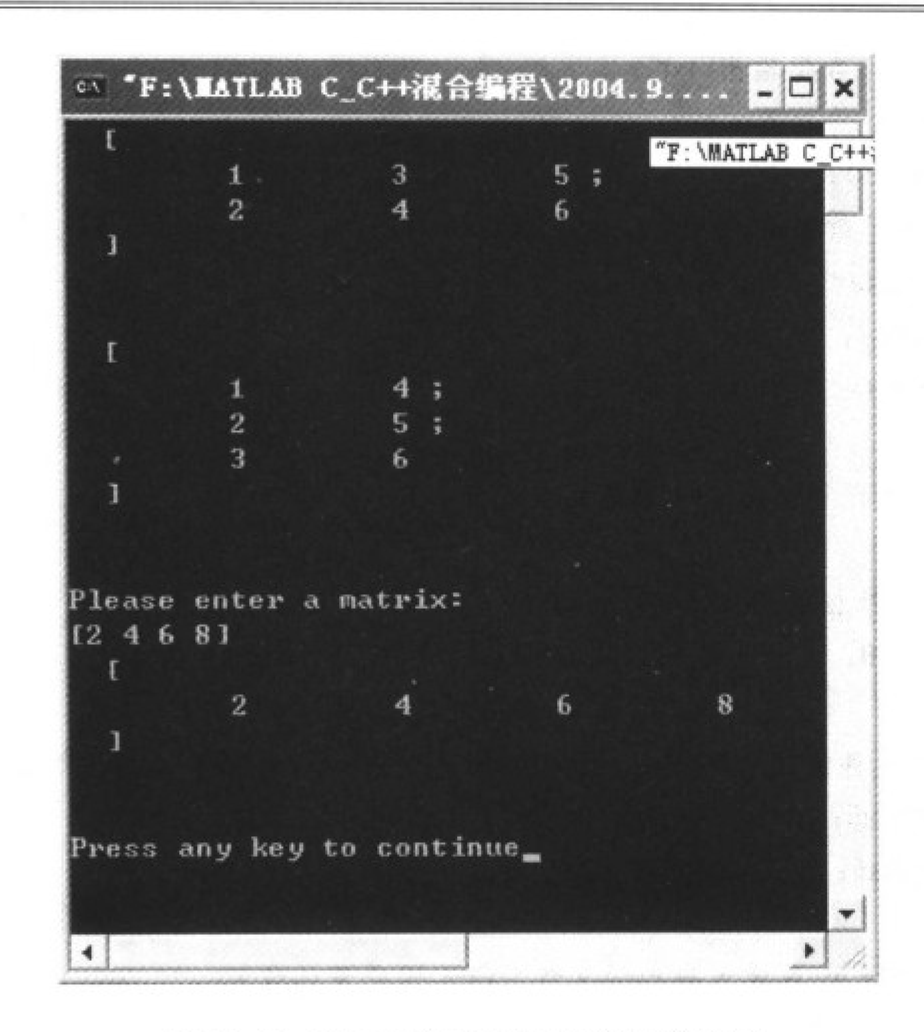


图 8-4　Matlab 输入输出实例运行结果

2. Matlab mwArray 阵列数据的输入和输出的其他方式

除了标准输入和输出函数以外，Matlab mwArray 阵列数据还有其他丰富的输入输出方式，例如：

① 采用文件流将变量输出到文件和从文件中读取变量。

② 通过 Matlab C++ 数学库自己的文件 I/O 实现变量的读写。

③ 采用 MAT 文件导入和导出数据。

④ 将 mwArray 数据输出到字符串中和从字符串中读取 mwArray 阵列数据。

到底选用何种 Matlab mwArray 阵列数据的输入和输出方式，需要根据用户使用 Matlab C++ 数学库的方式来决定。例如在控制台模式下，使用标准输入/输出方式比较简单方便；在 Visual C++ 6.0 MFC 工程中，使用文件输入/输出和字符串输入/输出则比较方便。下面的例子分别给出了上述 4 种 Matlab mwArray 阵列数据的输入/输出方式的实现方式。

```
/******************* mwArrayIO.cpp 文件 ******************/
#include "stdafx.h"
#include "matlab.hpp"
#include "mex.h"
#include "matrix.h"
#include "fstream.h"

int main(int argc, char * argv[])
{
    //采用文件流将变量输出到文件
```

```
char filename[]="数据输出.txt";
mwArray m_data;
m_data=rand(3,3);
ofstream out_file("数据输出.txt", ios::out);
out_file<<m_data<<ends;
m_data=magic(4);
out_file<<m_data<<ends;
out_file.close();
//采用文件流将存储在文件中的变量读入
mwArray A,B;
ifstream in_file("数据输出.txt", ios::in);
in_file>>A>>B;

cout<<"采用文件流从文件 "<<filename<<" 中读入的变量如下:"<<endl;
cout<<A<<endl;
cout<<B<<endl;

//通过 Matlab C++ 数学库自己的文件 I/O 实现变量的读写
char filename1[]="数据输出 1.txt";
mwArray mwfile(filename1);
mwArray fd=fopen(mwfile,"w");
fprintf(fd,"%f %f %f\n",A);
fprintf(fd,"\n");
fprintf(fd,"%4d %4d %4d %4d\n",B);
fclose(fd);

mwArray szA,szB,C,D;
szA=horzcat(3,3);
szB=horzcat(4,4);

fd=fopen(mwfile,"r");
C=fscanf(fd,"%f %f %f\n",szA);
D=fscanf(fd,"%4d %4d %4d %4d\n",szB);

cout<<"采用文件 I/O 从文件 "<<filename1<<" 中读入的变量如下:"<<endl;
cout<<C<<endl;
cout<<D<<endl;

fclose(fd);
//字符型 mwArray 对象数据通过文件 I/O 的读写
char filename2[]="字符串输出.txt";
mwArray s1("实践是检验真理的惟一标准!");
mwArray s2("没有付出,就没有收获!");
```

```
    mwArray s3,s4;
    mwArray mwfile1(filename2);
    fd = fopen(mwfile1,"wr");
    fprintf(fd,"%s\n",s1,s2);
    fclose(fd);
    fd = fopen(mwfile1,"r");
    s3 = fgetl(fd);
    s4 = fgetl(fd);
    cout<<"从文件"<<filename2<<"中读入的字符串变量如下:"<<endl;
    cout<<s3<<endl;
    cout<<s4<<endl;
    fclose(fd);
    //采用 MAT 文件导入和导出数据
    char filename3[] = "dataoutput.mat";
    save(filename3,"A",A,"B",B,"s1",s1,"s2",s2);
    mwArray a1,b1,c1,d1;
    load(filename3,"A",&a1,"B",&b1,"s1",&c1,"s2",&d1);
    if((tobool(A == a1))&&
        (tobool(B == b1))&&
        (tobool(s1 == c1))&&
        (tobool(s2 == d1)))
    {
        cout<<"采用 MAT 文件进行数据导入和导出正确！\n"<<endl;
    }
    else
    {
        cout<<"采用 MAT 文件进行数据导入和导出错误！\n"<<endl;
    }
    //采用将 mwArray 对象数据输出到字符串中
    mwArray mwstr;
    mwstr = sprintf("%6.4f %6.4f %6.4f;\n",A);
    cout<<"将对象 A 的数据输出到字符串中:"<<endl;
    cout<<mwstr<<endl;
    return 0;
}
```

程序运行结果如下所示。

```
采用文件流从文件 数据输出.txt 中读入的变量如下:
  [
    0.95013    0.48598    0.45647 ;
    0.23114    0.89130    0.01850 ;
    0.60684    0.76210    0.82141
  ]
```

```
[
        16        2        3        13 ;
         5       11       10         8 ;
         9        7        6        12 ;
         4       14       15         1
]
采用文件 I/O 从文件 数据输出 1.txt 中读入的变量如下：
[
    0.95013    0.48598    0.45647 ;
    0.23114    0.89130    0.01850 ;
    0.60684    0.76210    0.82141
]
[
        16        2        3        13 ;
         5       11       10         8 ;
         9        7        6        12 ;
         4       14       15         1
]
从文件 字符串输出.txt 中读入的字符串变量如下：
'实践是检验真理的惟一标准！'
'没有付出，就没有收获！'
采用 MAT 文件进行数据导入和导出正确！
将对象 A 的数据输出到字符串中：
'0.9501 0.2311 0.6068;
0.4860 0.8913 0.7621;
0.4565 0.0185 0.8214;
'
```

8.3.2 操作 Matlab mwArray 阵列

所有的 Matlab C++数学库函数的输入都是 mwArray 类型的 Matlab 阵列，因而熟悉 Matlab mwArray 阵列的操作是使用 Matlab C++ 数学库的基础。mwArray 支持的 Matlab 阵列类型有：

- 数值型阵列
- 稀疏矩阵阵列
- 字符型阵列
- 元组阵列
- 结构体阵列

其中，mwArray 只支持将二维数值型阵列转化为稀疏矩阵阵列。

1. 创建 Matlab mwArray 阵列的操作

创建 Matlab mwArray 阵列对象有两种方式，一种是通过 mwArray 类的构造函数创建 Matlab mwArray 阵列；另一种是其他函数如 rand 等的返回直接创建 Matlab mwArray 阵列。

常用的创建 Matlab mwArray 阵列对象的方法如表 8-1 所列。

表 8-1　创建 Matlab mwArray 阵列对象的方法

操作代码	功能说明
mwArray A;	直接创建一个未初始化的 mwArray 对象
empty()	创建一个空的 Matlab 阵列([])
mwArray(double)　//从 double 型变量创建 mwArray(int)　//从 int 型变量创建	创建一个标量数值阵列(只有一个元素)

mwArray(int M,int N,double * data);
mwArray(int M, int N, int * data);
mwArray(int M, int N, int * data);
	创建 M×N 行数值阵列(即矩阵)

mwArray(mxArray *);　//指针模式 mwArray(const mxArray &);　//引用模式	复制已经存在的 mwArray 类对象
mwArray(int start, int distance, int end) mwArray(double start,double distance ,double end) 相当于 matlab 的 start:distance:end	创建一个线性变化的数值阵列
mwArray(const mwSubArray&)	从索引数组中产生 mwArray 对象
采用两个函数:horzcat()和 vertcat() 其中,horzcat(A,B)和 vertcat(A,B)相当于 Matlab 的[A B]和[A;B]语句	通过连接现有的 mwArray 阵列对象产生新的 Matlab mwArray 阵列对象
ones()、zeros()、rand()和 randn()	创建全 1、全零、或者均匀分布的随机数和高斯分布的随机数构成的 Matlab 阵列
eye()和 magic()	创建单位矩阵和魔方矩阵

下面给出创建 Matlab mwArray 阵列对象几种常用方式的实现方法及程序运行的结果。

```
/****************mwArrayCreate.cpp 文件******************/
#include "stdafx.h"
#include "matlab.hpp"
#include <stdlib.h>
#ifdef GCC
#ifndef EXIT_SUCCESS
  #define EXIT_SUCCESS 0
#endif
#endif
double data[]={1,2,3,4,5,6,7,8};
int main(int argc, char * argv[])
{
    //创建二维数组
    mwArray mwadata(2,4,data);          //从原有的 double 型数组中创建
    mwArray copymwdata(mwadata);        //复制原有的 mwArray 对象
    //从 C double 型数组中创建按列存储方式的 Matlab mwArray 阵列
```

```
mwArray mwadataRowMajor = row2mat(2,4,data);
//创建多维数组
//创建二维数组,然后采用 reshape 来实现
mwArray multiD;
multiD = reshape(mwadata,2,2,2);
//采用 Matlab C++数学库的函数实现,如 rand,zeros,randn 等
//Matlab 语句 rand333 = rand(3,3,3)eye33 = eye(3,3)
mwArray rand333,eye33,assignArray;
rand333 = rand(3,3,3);
eye33 = eye(3,3);
//直接采用"="来实现
/* Matlab 语句:
    for i= 1:3
        assignArray(i,i,i) = 1;
    end
*/
int i=0;
for(i=0;i<3;i++)
{
    assignArray(i+1,i+1,i+1) = 1;
}
//Matlab 的":"运算符在 Matlab C++数学库中的实现用 ramp
//Matlab 语句   rampindex = 1:3:20;colonindex = 0:2:20
mwArray rampindex = ramp(1,3,20);
mwArray colonindex = colon(0,2,20);
//Matlab 语句   D = zeros(3,3,3);D(:,:,3) = 5
mwArray D;
D = zeros(3,3,3);
D(colon(),colon(),3) = 5;
//采用横向或者纵向连接新的数组元组的方式
//Matlab 语句   A = [1 2 3];B = [4 5 6];C = [A;B];
mwArray A,B,C;
A = horzcat(1,2,3);
B = horzcat(4,5,6);
C = vertcat(A,B);
cout<<" mwadata = "<<mwadata<<endl;
cout<<" copy of mwadata = "<<mwadata<<endl;
cout<<"采用 row2mat 从按行存储的 C double 数组中创建按列存储的 Matlab mwArray 阵列"<<endl;
cout<<"通过通配符 colon 实现赋值:"<<mwadataRowMajor<<endl;
cout<<"通过 reshape 创建的 2×2×2 数组 = "<<multiD<<endl;
cout<<"3×3×3 随机数数组 = "<<rand333<<endl;
cout<<"3×3 单位矩阵 = "<<eye33<<endl;
cout<<"直接通过 = 号创建的 3×3×3 维数组 = "<<assignArray<<endl;
```

```
    cout<<"通过 ramp 创建的索引数组="<<rampindex<<endl;
    cout<<"通过 colon 创建的索引数组="<<colonindex<<endl;
    cout<<"D="<<D<<endl;
    cout<<"横向连接产生的数组 A=\t"<<A<<"B=\t"<<B<<endl;
    cout<<"A 和 B 垂向连接产生的数组 C="<<C<<endl;
    return EXIT_SUCCESS;
}
```

程序执行结果如下所示。

```
mwadata =    [
        1        3        5        7 ;
        2        4        6        8
  ]
copy of mwadata =   [
        1        3        5        7 ;
        2        4        6        8
  ]
采用 row2mat 从按行存储的 C double 数组中创建按列存储的 Matlab mwArray 阵列
通过通配符 colon 实现赋值： [
        1        2        3        4 ;
        5        6        7        8
  ]
通过 reshape 创建的 2×2×2 数组=[
(:,:,1)=
  [
        1        3 ;
        2        4
  ]
(:,:,2)=
  [
        5        7 ;
        6        8
  ]
3×3×3 随机数数组=[
(:,:,1)=
  [
    0.95013    0.48598    0.45647 ;
    0.23114    0.89130    0.01850 ;
    0.60684    0.76210    0.82141
  ]
(:,:,2)=
  [
    0.44470    0.92181    0.40571 ;
    0.61543    0.73821    0.93547 ;
    0.79194    0.17627    0.91690
```

```
  ]
(:,:,3) =
  [
    0.41027    0.35287    0.13889 ;
    0.89365    0.81317    0.20277 ;
    0.05789    0.00986    0.19872
  ]
3×3 单位矩阵 =   [
          1          0          0 ;
          0          1          0 ;
          0          0          1
  ]
直接通过=号创建的 3×3×3 维数组=[
(:,:,1) =
  [
          1          0          0 ;
          0          0          0 ;
          0          0          0
  ]
(:,:,2) =
  [
          0          0          0 ;
          0          1          0 ;
          0          0          0
  ]
(:,:,3) =
  [
          0          0          0 ;
          0          0          0 ;
          0          0          1
  ]
通过 ramp 创建的索引数组 =   [
          1          4          7          10          13          16          19
  ]
通过 colon 创建的索引数组 =   [
          0          2          4          6          8          10          12          14
                     16          18          20
  ]
D=[
(:,:,1) =
  [
          0          0          0 ;
          0          0          0 ;
          0          0          0
  ]
```

```
(:,:,2) =
 [
        0       0       0 ;
        0       0       0 ;
        0       0       0
 ]
(:,:,3) =
 [
        5       5       5 ;
        5       5       5 ;
        5       5       5
 ]
横向连接产生的数组 A=      [
        1       2       3
 ]
B=      [
        4       5       6
 ]
A和B垂向连接产生的数组 C=  [
        1       2       3 ;
        4       5       6
 ]
```

2. Matlab C++数学库对稀疏矩阵的操作

当矩阵中非零元素占所有元素的比例很低的时候，采用稀疏矩阵可以有效地降低矩阵占用的存储空间。表 8-2 给出了 Matlab 稀疏矩阵常用的操作函数。

表 8-2 操作 Matlab C++数学库的稀疏矩阵

操作函数	功能说明
sparse()	创建 sparse 数组
full()	sparse 数组与正常数组的转换
spones()	将 sparse 数组中的非零元素全部转换为 1
sprand()、sprandn()和 sprandnsym()	将 sparse 数组中的非零元素全部转换为随机数
spconvert()	将文本文件转换为稀疏矩阵
speye()	创建一个单位稀疏矩阵
nnz()	确定数值矩阵中的非零元素个数
nzmax()	判断稀疏矩阵中最多可以存储多少个非零元素
nonzeros()	返回包含矩阵中所有非零元素的向量

下面的例子给出了对 Matlab mwArray 稀疏矩阵的基本操作方法及从文本文件中将数据读到 Matlab mwArray 中的方法。

```
/****************** sparseArrayCreate.cpp 文件 ******************/
#include "stdafx.h"
```

```
#include "matlab.hpp"
#include <stdlib.h>
#ifdef GCC
    #ifndef EXIT_SUCCESS
    #define EXIT_SUCCESS 0
#endif
#endif
int main(int argc, char * argv[])
{
    //创建稀疏矩阵
    //直接从原来矩阵的基础上生成稀疏矩阵
    mwArray A,sparseA;
    A=eye(5,5);
    sparseA=sparse(A);
    //利用索引创建 Matlab 矩阵
    double inums[]={3,4,5,4,5,6};
    double jnums[]={4,3,3,5,5,4};
    mwArray indexS;
    mwArray i(1,6,inums,NULL);
    mwArray j(1,6,jnums,NULL);
    indexS=sparse(i, j, 9, 8, 7);
    //从文本文件中读取数据并将其转换为稀疏矩阵
    //注意 data 阵列的数据格式:
                        行   列      数据
                        8    1       6.0000
                        3    5       7.0000
                        4    9       2.0000
                        9    9       1.8000
    mwArray filename("sparsedata.dat");
    mwArray data ;
    mwArray spdata ;
    mwArray fid;
    fid=fopen(filename,"r");
    data=fscanf(fid,"%f",mwArray(3 * 4));
    data=reshape(data,3,4);
    data=transpose(data);
    cout<<data<<endl;
    spdata=spconvert(data);
    fclose(fid);
    cout<<"单位矩阵 A:"<<A<<endl;
    cout<<"单位矩阵 A 的稀疏矩阵"<<sparseA<<endl;
    cout<<"利用索引创建的稀疏矩阵"<<indexS<<endl;
    cout<<"将上述稀疏矩阵转换为一般矩阵形式:"<<full(indexS)<<endl;
```

```
    cout<<"分析稀疏矩阵的属性:"<<endl;
    cout<<"\t 非零元素个数:"<<nnz(indexS)<<endl;
    cout<<"\t 最大可存储的非零元素个数:"<<nzmax(indexS)<<endl;
    cout<<"从文件读取的稀疏矩阵为 :"<<spdata<<endl;
    return 0;
}
```

程序执行结果如下所示。

```
单位矩阵 A: [
        1       0       0       0       0 ;
        0       1       0       0       0 ;
        0       0       1       0       0 ;
        0       0       0       1       0 ;
        0       0       0       0       1
  ]
单位矩阵 A 的稀疏矩阵
ans =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
利用索引创建的稀疏矩阵
ans =
   (4,3)        9
   (5,3)        9
   (3,4)        9
   (6,4)        9
   (4,5)        9
   (5,5)        9
将上述稀疏矩阵转换为一般矩阵形式: [
        0       0       0       0       0       0       0 ;
        0       0       0       0       0       0       0 ;
        0       0       0       9       0       0       0 ;
        0       0       9       0       9       0       0 ;
        0       0       9       0       9       0       0 ;
        0       0       0       9       0       0       0 ;
        0       0       0       0       0       0       0 ;
        0       0       0       0       0       0       0
  ]
分析稀疏矩阵的属性:
    非零元素个数: [
        6
```

```
]
  最大可存储的非零元素个数： [
        6
]
从文件读取的稀疏矩阵为 ：
ans =
   (8,1)        6.0000
   (3,5)        7.0000
   (4,9)        2.0000
   (9,9)        1.8000
```

3. Matlab C++数学库对字符型阵列的操作

在 Matlab 及 Matlab C++数学库中，一维字符阵列与普通 C/C++语言的字符串相对应，二维字符阵列与普通的 C/C++语言的字符串数组相对应。对于 Matlab 二维字符阵列而言，每行(即字符串数组中的每个字符串)的长度必须一致。因此，Matlab 二维字符阵列每行的长度必须是最长字符串的长度。

Matlab C++数学库中关于字符型阵列的主要函数如表 8-3 所列。

表 8-3 字符阵列的常用操作函数及其功能

操作函数	功能说明
mwArray string("abcd")； /构造函数	创建字符阵列
char_func()	由数值阵列创建字符阵列
double_func()	将字符阵列转换为其 ASCII 码值字符阵列
str2mat,strcat 和 strhcat	将一维字符阵列连接为多维字符阵列，多维字符阵列中小于最大长度字符串长度的行用空格扩展
cellstr()	从字符阵列中生成元组阵列，其中字符阵列的每一行作为一个元组阵列的元素
num2str	将数值阵列的数字按照一定的格式转换为相应字符表示形式
str2num	与 num2str 相反，将用字符串表示的数字转换为数值阵列
int2str	将输人的数值阵列四舍五人转换为整数，然后将相应的整数值转换为字符串

下面的实例给出了 Matlab C++ 数学库对字符阵列常用的操作方法，其中包括：

① 创建字符阵列。

② 将存储 ASCII 码值的数值阵列转换为字符阵列。

③ 将一维字符阵列连接为多维字符阵列。

④ 将字符阵列转换为元组阵列表示形式。

⑤ 从字符阵列中提取数值阵列的数据。

```
/*****************charArrayCreate.cpp**********************/
#include "stdafx.h"
#include "matlab.hpp"
#include <stdlib.h>
#ifdef GCC
```

```
#ifndef EXIT_SUCCESS
  #define EXIT_SUCCESS 0
#endif
#endif
int main(int argc, char * argv[])
{
    //创建字符型数组
    mwArray sHello("Hello!");                     //利用构造函数
    //将其他数值类型数组转换为字符形式
    mwArray doubleData,sFromDoubleData;
    doubleData=horzcat(109,121,32,115,116,114,105,110,103);
    sFromDoubleData=char_func(doubleData);        //doubleData 中保存字符的 ASCII 码值
    //用一维字符串重新构造字符串和字符矩阵
    mwArray sA=char_func(sHello,sFromDoubleData); //采用 char_func 函数将字符串组合为字符矩阵
    mwArray sB=horzcat(sHello,sFromDoubleData);   //用连接函数构造更长的字符串
    mwArray sC=str2mat(sHello,sFromDoubleData);   //采用 str2mat 函数构造字符矩阵
    mwArray sCell=cellstr(sC);                    //将字符矩阵转换为元组表示形式
    //将字符类型的数值转换为数值表示形式
    mwArray sS("5.1234,4.36,6.45");
    mwArray sN=str2num(sS);
    cout<<"利用构造函数创建字符数组:"<<sHello<<endl;
    cout<<"数值类型数组:"<<doubleData<<endl;
    cout<<"将其转换为字符:"<<sFromDoubleData<<endl;
    cout<<"采用 char_func 函数将字符串组合为字符矩阵:"<<sA<<endl;
    cout<<"采用连接函数将字符串连接为一个长字符串:"<<sB<<endl;
    cout<<"采用 str2mat 函数将字符串连接为一个长字符串:"<<sC<<endl;
    cout<<"字符矩阵的元组表示形式:"<<sCell<<endl;
    cout<<"字符串:"<<sS<<"转换为数值:"<<sN<<endl;
    return 0;
}
```

程序执行结果如下所示。

```
利用构造函数创建字符数组:'Hello! '
数值类型数组: [
      109      121      32      115      116      114      105      110      103
  ]
将其转换为字符:'my string'
采用 char_func 函数将字符串组合为字符矩阵: [
    'Hello!    ' ;
    'my string' ;
  ]
采用连接函数将字符串连接为一个长字符串:'Hello! my string'
采用 str2mat 函数将字符串连接为一个长字符串:[
```

```
    'Hello!     ';
    'my string';
  ]
字符矩阵的元组表示形式：    'Hello!  '
    'my string'
字符串:'5.1234,4.36,6.45' 转换为数值： [
    5.12340   4.36000   6.45000
  ]
```

4. Matlab C++数学库对元组阵列的操作

采用 Matlab 元组阵列可以将不同类型的 Matlab 阵列组织到一起，Matlab C++数学库中同样提供了对 Matlab 元组阵列的操作。其中常用的函数如表 8-4 所列。

表 8-4 元组阵列的常用操作函数及其功能

操作函数	功能说明
cell	创建 Matlab 元组阵列
cellstr	将 Matlab 字符阵列转换为元组阵列，见本书 Matlab C++数学库的字符阵列操作部分
cellhcat	通过连接现有的 Matalb 阵列，创建一个元组阵列
struct2cell	将结构体转换为元组阵列表示
num2cell	将数值阵列转换为元组阵列表示
celldisp	在标准输出设备上显示元组阵列的内容

下面的实例给出了 Matlab C++ 数学库对元组阵列常用的操作方法，其中包括：

① 采用各种方法创建元组阵列。

② 直接为元组阵列元素赋值。

③ 将数值阵列转换为元组阵列。

④ 采用 celldisp 函数显示元组阵列。

```
/*******************cellArrayCreate.cpp 文件****************/
#include "stdafx.h"
#include "matlab.hpp"
#include <stdlib.h>
#ifdef GCC
  #ifndef EXIT_SUCCESS
  #define EXIT_SUCCESS 0
  #endif
#endif
int main(int argc, char * argv[])
{
    //创建 Cell 阵列，并采用 cell 创建函数
    mwArray cCell;
    cCell = cell(3,3);
```

```
    //数值阵列到 Cell 阵列转换函数
    double data[9]={1,2,3,4,5,6,7,8,9};
    mwArray mdata(3,3,data);
    mwArray transferCell;
    //将 data 数组的每列转换为一个 cell 元素
    transferCell=num2cell(mdata,mwArray(1));
    //连接现有阵列
    //Matlab 语句 hC={ 'jon' ones(2) magic(3) 5 }vC={ 'jon'; ones(2); magic(3); 5 }
    mwArray hC=cellhcat("jone",ones(3),magic(3),4);
    mwArray vC=vertcat(cellhcat("jone"),cellhcat(ones(3)), cellhcat(magic(3)),cellhcat(4));
    //直接为 Cell 阵列赋值
    //Matlab 语句 aD{2,2}='hello! '
    //aD.cell(2,2)中的 cell 函数相当于{},与全局函数 cell 不同
    mwArray aD;
    aD.cell(2,2)=("hello");
    cout<<"采用 cell(3,3)创建的 Cell 阵列:\n"<<cCell<<endl;
    cout<<"采用 num2cell 创建的 Cell 阵列:\n"<<transferCell<<endl;
    cout<<"采用 cellhcat 创建的 Cell 阵列:\n"<<hC<<endl;
    cout<<"采用 vertcat+cellhcat 创建的 Cell 阵列:\n"<<vC<<endl;
    cout<<"采用直接赋值的方法创建的 Cell 阵列:\n"<<aD<<endl;
    //采用 celldisp 显示 Cell 阵列的内容
    cout<<"采用 celldisp 显示 Cell 阵列的内容:"<<endl;
    celldisp(vC,"vC");
    return 0;
}
```

程序执行结果如下所示。

```
采用 cell(3,3)创建的 Cell 阵列:
    []      []      []
    []      []      []
    []      []      []
采用 num2cell 创建的 Cell 阵列:
   [1]     [1]     [1]
   [1]     [1]     [1]
   [1]     [1]     [1]
采用 cellhcat 创建的 Cell 阵列:
    'jone'     [3×3 double]     [3×3 double]     [4]
采用 vertcat+cellhcat 创建的 Cell 阵列:
    'jone'
    [3×3 double]
    [3×3 double]
    [           4]
采用直接赋值的方法创建的 Cell 阵列:
```

```
      []          []
      []       'hello'
```

采用 celldisp 显示 Cell 阵列的内容：

```
vC{1} =
jone
vC{2} =
      1     1     1
      1     1     1
      1     1     1
vC{3} =
      8     1     6
      3     5     7
      4     9     2
vC{4} =
      4
```

5. Matlab C++数学库对结构阵列的操作

Matlab 结构体阵列的主要操作函数如表 8-5 所列。

表 8-5 结构体阵列的常用操作函数及其功能

操作函数	功能说明
struct_func()	创建并初始化 Matlab 结构体阵列
cell2struct()	将 Matlab 元组阵列转换为结构体阵列
fieldnames()	返回 Matlab 结构体阵列的域名
isfield()	判断输入的字符串是否为结构体阵列的域名
getfield()	根据域名得到 Matlab 结构体阵列的内容
setfield()	根据域名设置 Matlab 结构体阵列的内容
rmfield()	根据域名去除结构体阵列的域

下面的实例给出了 Matlab C++数学库构造结构体阵列的不同方法，以及采用isfield 函数根据结构体域名查询结构体域是否存在的方法。具体代码如下所示。

```
/* * * * * * * * * * * * * * * * * * structArrayCreate.cpp 文件 * * * * * * * * * * * * * * * * */
#include "stdafx.h"
#include "matlab.hpp"
#include <stdlib.h>
#ifdef GCC
  #ifndef EXIT_SUCCESS
  #define EXIT_SUCCESS 0
  #endif
#endif
int main(int argc, char * argv[])
{
```

```
    //构建 Matlab 结构阵列
    mwArray sA;
    sA = struct_func("Name","Daniel",
                     "Phone",888888,
                     "Address","北京市外国语学院");      //结构体域名  结构体域值
    //采用 cell2struct 转换函数构建结构阵列
    /* Matlab 语句
        sB = {'Daniel',888888,'北京市外国语学院'};
        sC = {'Name','Phone','Address'};
        sD = cell2struct(sB,sC,2);
    */
    mwArray sB,sC,sD;
    sB = cellhcat("Daniel",888888,"北京市外国语学院");
    sC = cellhcat("Name","Phone","Address");
    sD = cell2struct(sB,sC,2);     //注意 2
    /* Matlab 语句
        sB = {'Daniel';888888;'北京市外国语学院'};
        sC = {'Name','Phone','Address'};
        sD = cell2struct(sB,sC,1);
    */
    sB = vertcat(cellhcat("Daniel"),cellhcat(888888),cellhcat("北京市外国语学院"));
    sC = cellhcat("Name","Phone","Address");
    sD = cell2struct(sB,sC,1);     //注意 1
    cout<<"采用 struct_fucn 函数构建结构:\n"<<sA<<endl;
    cout<<"采用 cell2struct 转换函数构建结构:\n"<<sD<<endl;
    char str[30];
    cout<<"请输入需要查询的域名:\n"<<endl;
    cin >> str;
    mwArray is = isfield(sD,str);
    if(tobool(is))                 //tobool 用于将 mwArray 布尔变量转换为 C++布尔变量
    {
        cout<<"输入域名在结构体 sD 中存在! \n"<<endl;
    }
    return 0;
}
```

程序执行结果如下所示。

```
采用 struct_fucn 函数构建结构:
Name: 'Daniel'
Phone: 888888
Address: '北京市外国语学院'
采用 cell2struct 转换函数构建结构:
Name: 'Daniel'
```

```
Phone：888888
Address：'北京市外国语学院'
请输入需要查询的域名:Name
输入域名在结构体 sD 中存在!
```

6. 访问 mwArray 类中的数据

Matlab C++数学库提供可以访问 mwArray 类的内部数据的函数，这些函数都是 Matlab C++数学库的内部函数。通过这些函数，开发人员可以和其他外部开发环境如 Matlab C 数学库进行数据交互。

(1) GetData()函数

GetData()函数返回一个 mxArray 结构的指针，通过这个 mxArray 结构，开发人员可以通过 Matlab C 数学库的函数进行数据访问操作。使用 GetData()函数需要注意的是，不要释放 GetData()函数返回的 mxArray 指针指向的内存及 mxArray 中包含的数据。

(2) ExtractScalar()函数和 ExtractData()函数

相对于 GetData()函数来说，ExtractScalar()和 ExtractData()两个函数使用起来更安全，但是速度稍慢。ExtractScalar()函数用于从数值类型的 mwArray 对象中提取单个数据元素，ExtractData()函数则用来从数值类型的 mwArray 对象中提取多个数据元素。

(3) ToString()函数

使用 ToString()函数可以将字符类型 mwArray 中的字符数据转换为 C/C++语言字符串数据。例如：

```
mwArray A="Hello World!";
mwString s=A.ToString();
char *c=strdup((char *)s);
```

需要注意的是，字符指针 c 的生命周期与 mwString 相同，mwString 析构的时候指针 c 指向的内容会自动被释放，因而如果开发人员释放指针 c 指向的内容，就会在 mwString 析构的时候造成程序崩溃。如果开发人员需要对字符串 c 进行下一步的操作，可以先对其进行复制，然后再对复制的字符串进行操作。

```
/****************mwDataAccess.cpp 文件******************/
#include "stdafx.h"
#include "matlab.hpp"
#include "mex.h"
#include "matrix.h"
int main(int argc, char *argv[])
{
    mwArray A,B;
    mxArray *pMx=NULL;
    double *pData=NULL;
    int j,k;
    A=rand(3,3);
    pMx=A.GetData();
```

```
    pData = mxGetPr(pMx);
    cout<<"通过 GetData 和 Matlab C 数学库访问 mwArray 的数据:"<<endl;
    cout.precision(2);
    for(j = 0;j<9;j++)
    {
        cout<<pData[j]<<",";
    }
    cout<<endl;
    //构造复数数组 B
    B = A + rand(3,3) * i();
    cout<<"B = "<<endl;
    cout.precision(3);
    cout<<B<<endl;
    double rdata5,idata5, * prdata, * pidata;
    prdata = new double[3 * 3];
    pidata = new double[3 * 3];
    rdata5 = B.ExtractScalar(idata5,5);
    cout.precision(2);
    cout<<"B 的第 5 个元素为:"<<rdata5<<" + "<<idata5<<"i"<<endl;
    B.ExtractData(prdata,pidata);
    cout<<"通过 ExtractData 返回的数据遍历 B 的各个元素:"<<endl;
    for(j = 0;j<3;j++)
    {
        for(k = 0;k<3;k++)
        {
            cout<<prdata[3 * k + j]<<" + "<<pidata[3 * k + j]<<"i   ";
        }
        cout<<endl;
    }
    //ToString 的使用
    mwArray C = "No Pain! No Gain!";
    mwString sC = C.ToString();
    char * str = strdup((char * )sC);
    while( * str! = '\0')
    {
        cout<< * str;
        str++;
    }
    cout<<endl;
    return 0;
}
```

程序执行结果如下所示。

```
通过 GetData 和 Matlab C 数学库访问 mwArray 的数据：
0.95,0.23,0.61,0.49,0.89,0.76,0.46,0.019,0.82,
B =
  [
    0.95 + 0.44i    0.49 + 0.92i    0.46 + 0.41i ;
    0.23 + 0.62i    0.89 + 0.74i    0.02 + 0.94i ;
    0.61 + 0.79i    0.76 + 0.18i    0.82 + 0.92i
  ]
B 的第 5 个元素为：0.89 + 0.74i
通过 ExtractData 返回的数据遍历 B 的各个元素：
0.95 + 0.44i   0.49 + 0.92i   0.46 + 0.41i
0.23 + 0.62i   0.89 + 0.74i   0.019 + 0.94i
0.61 + 0.79i   0.76 + 0.18i   0.82 + 0.92i
No Pain! No Gain!
```

7. Matlab C++数学库对 Matlab 阵列操作常用的小技巧

使用 Matlab C++数学库的时候要尽量避免 Matlab 阵列和普通 C/C++语言数据结构之间的相互转换，以提高效率。但是，有时候这些转换又是不可避免的。在使用 Matlab C++数学库的过程中，容易出问题的地方就是 Matlab C++数学库和普通 C/C++语言的接口部分，这也是本章的重点。下面再给出几个 Matlab C++数学库 mwArray 类型和普通 C/C++语言的转换小技巧。

① 其他类型的数据向 mwArray 阵列转换，主要通过 mwArray 阵列的构造函数来完成。

② 确定 mwArray 阵列的大小，通过 size 函数完成。

③ BOOL 型数据转换，通过 tobool 函数完成。

为了更好地说明上述三个小技巧的实现方法，下面给出一个实例。

```
/* * * * * * * * * * * * * * * * * * mwDataChange.cpp 文件 * * * * * * * * * * * * * * * * * */
#include "stdafx.h"
#include "matlab.hpp"
#include "mex.h"
#include "matrix.h"
int main(int argc, char * argv[])
{
    //数组构建
    static double data[] = {1,3,5,7,9,11};
    double * pdata = NULL;
    mwArray A,B,C,D,E,F;
    mxArray * mxD = NULL;
    A = 5;
    B = "Hello World!";
    C = mwArray(2,3,data);
    mxD = mxCreateDoubleMatrix(2,3,mxREAL);
```

```
pdata = mxGetPr(mxD);
for(int i=0;i<6;i++)
{
    pdata[i] = i;
}
D = mxD;
E = ramp(1,10);
F = E(8);
//数组的大小获取
mwArray m,n;
int  m1,n1;
size(mwVarargout(m,n),C);         //方式 1
m1 = size(&n1,C);                 //方式 2,只适用于二维的情况
int  dims;
int  ndims[2];
dims = C.Size();
C.Size(ndims);                    //方式 3
//mwArray 布尔型变量转换为 C/C++布尔型变量
mwArray isCReal = isreal(C);
//C 变量数据指针的获取
mxArray * pCMxData = C.GetData();
double * pCData = mxGetPr(pCMxData);
cout<<"构造 1×1 的数值阵列:\n"<<A<<endl;
cout<<"构造字符阵列:\n"<<B<<endl;
cout<<"构造 2×3 的数值阵列:\n"<<C<<endl;
cout<<"通过 mxArray 构造 2×3 的数值阵列:\n"<<D<<endl;
cout<<"通过 mwSubArray 构造数值阵列:\n"<<F<<endl;
cout<<"数组 C 的维数为:"<<dims<<endl;
cout<<"方法 1:\n"<<"行:"<<m<<"\n"<<"列:"<<n<<"\n"<<endl;
cout<<"方法 2:\n"<<"行:"<<m1<<"\n"<<"列:"<<n1<<"\n"<<endl;
cout<<"方法 3:\n"<<"行:"<<ndims[0]<<"\n"<<"列:"<<ndims[1]<<"\n"<<endl;
cout<<"C 数组的元素个数为:"<<C.EltCount()<<"\n"<<endl;
if(tobool(isCReal))
{
    cout<<"C 数组的数组类型为实数!"<<endl;
}
else
{
    cout<<"C 数组的数组类型为复数!"<<endl;
}

cout<<"阵列 C 的内容如下:\n"<<endl;
for(i=0;i<C.EltCount();i++)
```

```
    {
        cout<<pCData[i]<<"    ";
    }
    cout<<endl;
    return 0;
}
```

程序执行结果如下所示。

```
构造 1×1 的数值阵列:
 [
         5
 ]
构造字符阵列:
'Hello World! '
构造 2×3 的数值阵列:
 [
         1          5          9 ;
         3          7          11
 ]
通过 mxArray 构造 2×3 的数值阵列:
 [
         0          2          4 ;
         1          3          5
 ]
通过 mwSubArray 构造数值阵列:
 [
         8
 ]
数组 C 的维数为:2
方法 1:
行: [
         2
 ]
列: [
         3
 ]
方法 2:
行:2
列:3
方法 3:
行:2
列:3
C 数组的元素个数为:6
```

```
C数组的数组类型为实数!
阵列C的内容如下:
1    3    5    7    9    11
```

8.3.3 调用系统函数

单输入/输出的函数调用是最简单的系统函数调用方法,此时调用 Matlab C++ 数学库的函数与普通 C/C++函数的调用方法相同。如调用正弦函数 y=sin(x)。

当输出参数大于一个时,需要将第二个输出参数的引用作为输入参数传递给调用函数。例如,对于这种形式的 Matlab 函数调用:[i,j,y1]=find(y>0.5);对应的 Matlab C++ 数学库的调用方式为:i=find(&j,&y1,y>0.5);

当输入和输出参数的个数大于 32 时,需要构造一个 mwVarargin 和 mwVarargout 对象。每个 mwVarargin 对象在构造的时候可以传递 32 参数,由于 mwVarargin 在构造的时候可以包含 mwVarargin 对象,因而理论上采用 mwVarargin 参数可以传递无穷多个输入参数。例如对于 vertcat,如果输入的参数个数为 90 个,需要采用下面的调用形式:

```
out = vertcat(
            mwVarargin(
                        mwVarargin(
                                    in1,in2…in32
                                ),
                        in33,in34…in63
                    ),
            in64…in90
           )
```

当传递输出参数个数大于 32 时,需要构造 mwVarargout 对象,mwVarargout 的构造方式与 mwVarargin 类似。例如利用 size 函数求数值阵列 X 的各维大小,当输出参数的个数大于 32 时,可以采用下面的调用方式:

```
size(
       mwVarargout(
                   mwVarargout(
                               out1,out2,out3,…,out32
                           ),
                   out33,out34,…,out60
               ),
       X
    );
```

下面的实例演示了如何处理不定个数输入参数及多输出参数的情况。

```
/* * * * * * * * * * * * * * * * * * * mwCallFunction.cpp 文件 * * * * * * * * * * * * * * * * * * */
#include "stdafx.h"
#include "matlab.hpp"
```

```
#include "matrix.h"
int main(int argc, char * argv[])
{
    //输入参数个数不定的情况
    //计算下三角阵
    mwArray x=vertcat(horzcat(1,2,3,4),horzcat(5,6,7,8),horzcat(9,10,11,12));
    mwArray trilx,trilx1;
    trilx=tril(x);
    trilx1=tril(x,-1);
    //多个输出参数的情况
    //查找数值阵列中大于0.5的元素
    mwArray y=rand(4,4);
    mwArray y1;
    mwArray i,j;
    mxArray *pi, *pj, *py, *py1;
    int  k;
    i=find(&j,&y1,y>0.8);
    pi=i.GetData();
    pj=j.GetData();
    py=y.GetData();
    py1=y1.GetData();
    cout<<"x=:\n"<<x<<endl;
    cout<<"tril(x)=:\n"<<trilx<<endl;
    cout<<"tril(x,-1)=:\n"<<trilx1<<endl;
    cout<<"随机数值阵列 y=:\n"<<y<<endl;
    if(mxIsLogical(py1))
    {
        cout<<"i=find(&j,&y1,y>0.8);\n 输出 y1 为逻辑型变量"<<endl;
    }
    cout<<"随机数值阵列大于0.9的元素为:"<<endl;
    cout.precision(4);
    for(k=0;k<mxGetNumberOfElements(pi);k++)
    {
        cout<<"\t<"
            <<*((double*)mxGetData(pi)+k)
            <<","
            <<*((double*)mxGetData(pj)+k)
            <<">:"
            <<*((double*)mxGetData(py)+k)
            <<endl;
    }
    return 0;
}
```

程序执行结果如下所示。

```
x=：
 [
        1        2        3        4 ;
        5        6        7        8 ;
        9        10       11       12
 ]
tril(x)=：
 [
        1        0        0        0 ;
        5        6        0        0 ;
        9        10       11       0
 ]
tril(x,-1)=：
 [
        0        0        0        0 ;
        5        0        0        0 ;
        9        10       0        0
 ]
随机数值阵列 y=：
 [
    0.95013    0.89130    0.82141    0.92181 ;
    0.23114    0.76210    0.44470    0.73821 ;
    0.60684    0.45647    0.61543    0.17627 ;
    0.48598    0.01850    0.79194    0.40571
 ]
i=find(&j,&y1,y>0.8);
输出 y1 为逻辑型变量
随机数值阵列大于 0.9 的元素为：
    <1,1>:0.9501
    <1,2>:0.2311
    <1,3>:0.6068
    <1,4>:0.486
```

附录一　动态链接库基础知识

通过本书各章的学习，读者会发现在C/C++和Matlab混合程序设计的各种方法中都有"动态链接库"身影的出现，比如通过Matlab编译器编译动态链接库供VC++调用、MEX文件实际上也是动态链接库、COM组件也以动态链接库的形式存在，等等。因而，本书增加了动态链接库基础知识作为附录，使读者能够更好地理解C/C++和Matlab混合程序设计的内容。

A.1　为什么使用动态链接库？

代码共享一直是软件开发者追逐的梦想，最初级的代码共享就是将编写的程序源代码（比如函数或者类）提供给其他程序员使用。源代码共享有很多问题，比如软件代码的知识产权难以得到有效的保护。另外，由于程序员都会有自己喜好的风格，有时发现对方提供的源代码不完全符合自己的编程习惯，因而对其进行更改，这样源代码的标准性和统一性就很难保证，最后很有可能出现版本混乱。解决这个问题的办法是只提供编译后的二进制库文件和程序的接口，最初使用静态链接库的方式提供，即使用者在所开发的程序中通过程序接口调用库函数，这些库函数只有在链接时才会真正连接到应用程序中。静态链接库解决源代码共享的一些问题，但并不完美。假设两个程序同时运行，并且都使用了相同的静态链接库，那么实际上静态链接库就要被加载到内存中两次，从而浪费内存空间；当然由于两个程序都链接了相同的静态链接库，肯定也会浪费硬盘空间（很长一段时间以来，硬盘空间的价格并不是像现在一样这么便宜）。这时候，就出现了动态链接的概念。动态链接库与静态链接库类似，都是以二进制文件的方式存在（在Windows的操作系统中是以dll为后缀的文件），所不同的是编译器在编译调用了动态链接库的程序时并不将库文件中的函数执行体链接到可执行文件中，而是只在可执行文件中保留一个函数调用的标记。当程序运行时，才由操作系统将动态链接库文件一并加载入内存，并映射到程序的地址空间中，这样就保证了程序能够正常调用到库文件中的函数。同时，当有多个调用动态链接库的程序运行时，也只有一份动态链接库的备份在内存中，即动态链接库在运行期是共享的。

使用动态链接库可以为程序开发带来如下优势：

● 动态链接库和用户程序可以分别开发，动态链接库和用户程序可以由不同的人在不同的地点完成，也可以使用不同的开发语言完成，只要符合标准，最终只要把动态链接库和用户程序放在一起使用即可。

● 动态链接库在编译链接时并不直接链接到应用程序中，这样就使目标程序比使用静态链接库时小，同时运行期又是共享动态链库，所以节省了磁盘存储空间和运行内存空间。

● 动态链接库动态加载的特点增加了程序的灵活性，可以实现诸如插件机制等功能。

A.2　C/C++语言实现动态链接库

动态链接库的编写方法与C/C++应用程序类似，这里首先介绍两组关键字：

1. _declspec(dllimport)

这组关键字表示从dll中导入函数或者数据，一般用在dll对应的头文件中。

2. _declspec(dllexport)

这组关键字表示从dll中导出函数或者数据，一般用在dll对应的工程文件中。

下面用一个简单的实例说明Windows下动态链接库的编写方法，首先打开Visual C++ 6.0，然后建立一个Win32 Dynamic-Link Library空工程，如图9-2所示。工程建立完毕以后，新建两个工程文件DLL_ADD.h和DLL_ADD.cpp，分别添加到工程中。其中DLL_ADD.h和DLL_ADD.cpp的源代码如下所示。

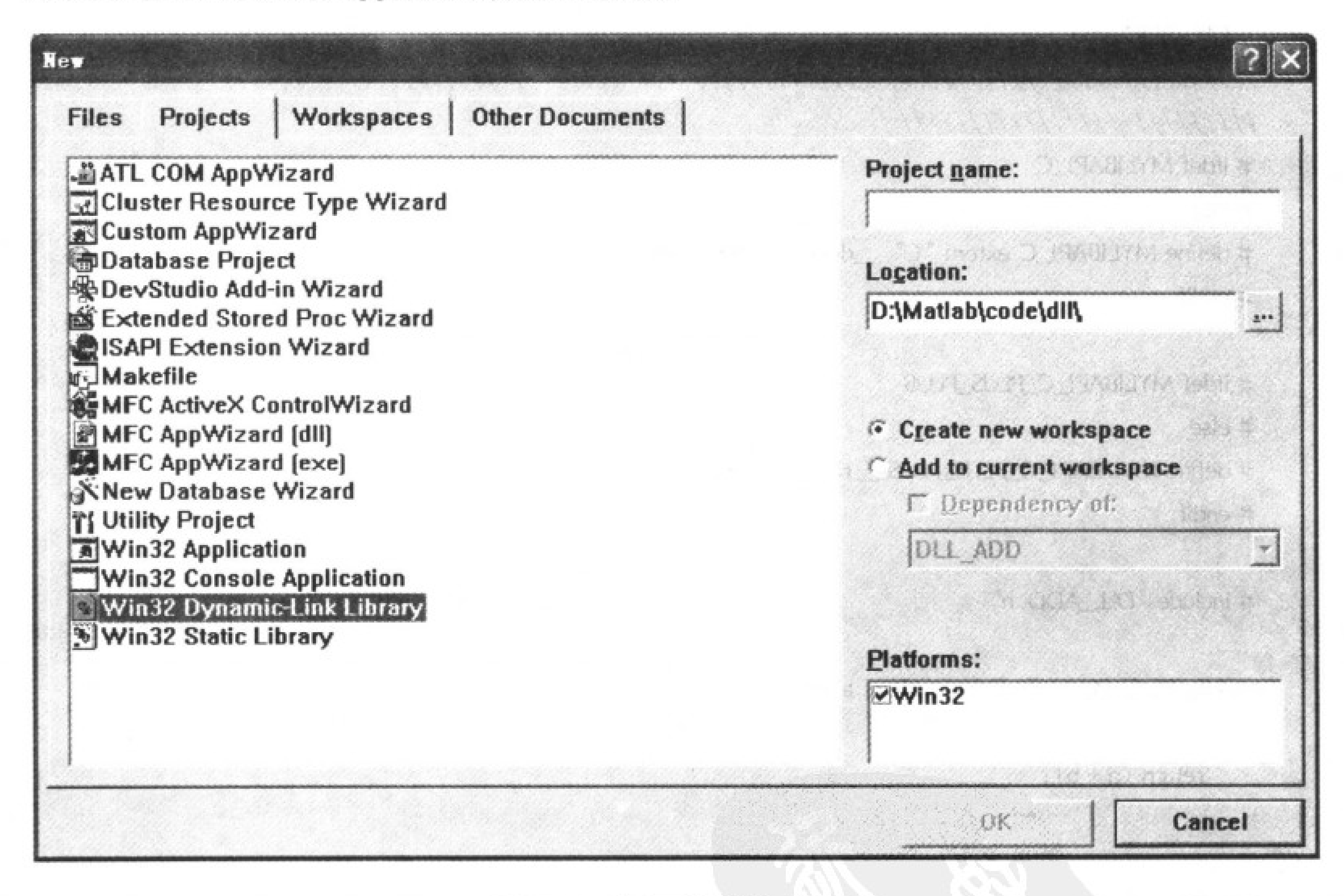


图9-1　建立动态链接库工程

```
//////////////////////////////////////////////////////////////////////////////////////
///////////////////////////
//DLL_ADD.h
//////////////////////////////////////////////////////////////////////////////////////
///////////////////////////
# ifdef MYLIBAPI_C
```

```
# else
    # define MYLIBAPI_C extern "C" __declspec(dllimport)
# endif

# ifdef MYLIBAPI_C_PLUS_PLUS
# else
# define MYLIBAPI_C_PLUS_PLUS __declspec(dllimport)
# endif

MYLIBAPI_C_PLUS_PLUS int Add(int a,int b);
MYLIBAPI_C int Sub (int a,int b);

//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
//DLL_ADD.cpp
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
# ifdef MYLIBAPI_C
# else
# define MYLIBAPI_C extern "C" __declspec(dllexport)
# endif

# ifdef MYLIBAPI_C_PLUS_PLUS
# else
# define MYLIBAPI_C_PLUS_PLUS __declspec(dllexport)
# endif

# include "DLL_ADD.h"

MYLIBAPI_C_PLUS_PLUS int Add(int a,int b)
{
    return (a+b);
}

MYLIBAPI_C int Sub(int a,int b)
{
    return (a-b);
}
```

编译上述工程即可得到动态链接库 DLL_ADD.dll(在 Debug 或 Release 目录下),此动态链接库实现了 Add 和 Sub 这两个函数,并且将其导出。采用 Visual C++ 6.0 提供的 Depends 工具(选择"开始"|"程序"|Microsoft Visual Studio 6.0 | Microsoft Visual Studio 6.0 Tools | Depends 启动),可以查看新生成的 DLL_ADD.dll 动态链接库。启动 Depends 以后,通过菜单打开

DLL_ADD. dll 文件，如图 9－2 所示，然后会得到其导出函数信息，如图 9－3 所示。

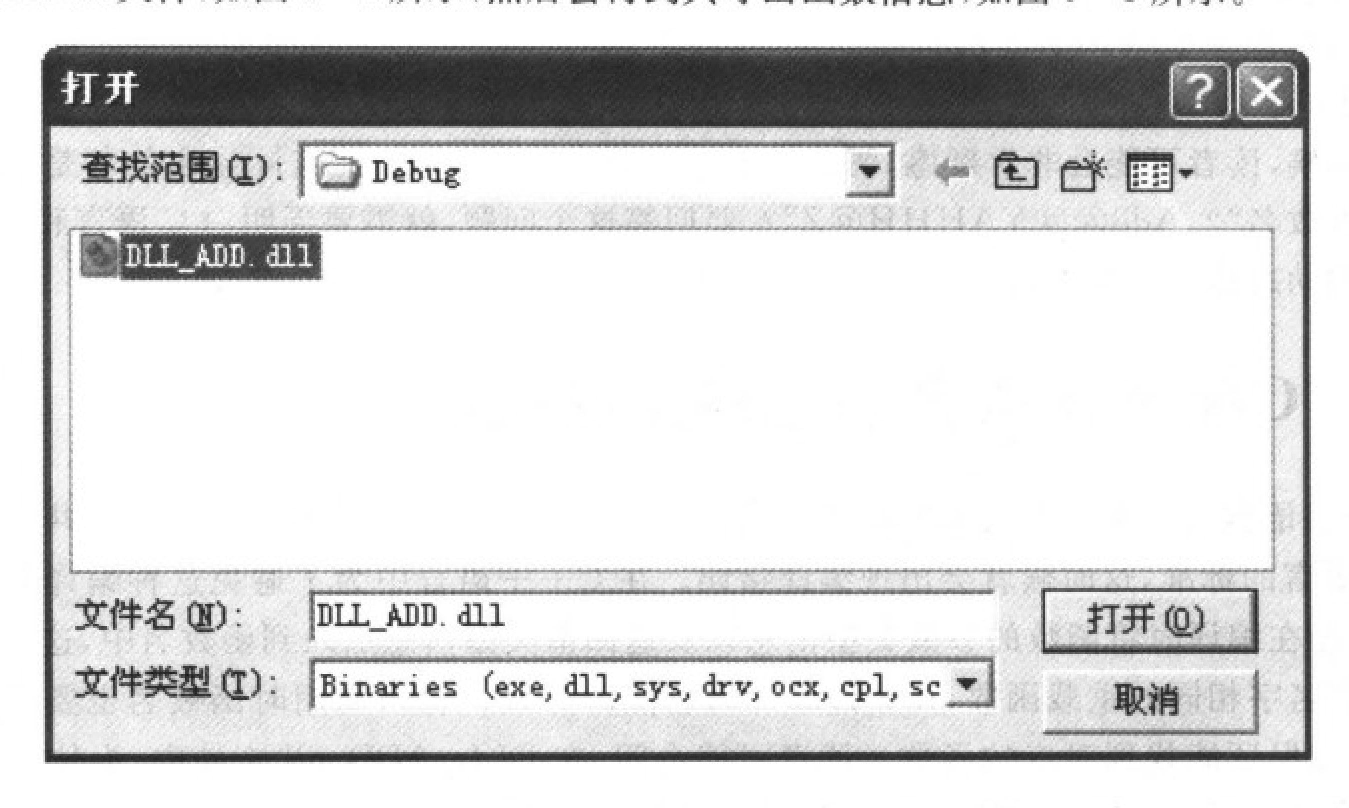


图 9－2　Depends 工具打开 DLL_ADD. dll 文件

Dependency Walker - [DLL_ADD. dll]

File Edit View Window Help

DLL_ADD. DLL
KERNEL32. DLL
NTDLL. DLL

Ordinal	Hint	Function	Entry Point
1 (0x0001)	0 (0x0000)	?Add@@YAHHH@Z	0x00001005
2 (0x0002)	1 (0x0001)	Sub	0x0000100A

Module	Time Stamp	Size	Attributes	Machine	Subsystem	Debug
DLL_ADD. DLL	07/23/07 2:21p	200,765	A	Intel x86	Win32 GUI	Yes
KERNEL32. DLL	04/16/07 11:54p	1,145,344	A	Intel x86	Win32 console	Yes
NTDLL. DLL	08/17/04 8:00p	591,360	A	Intel x86	Win32 console	Yes

For Help, press F1

图 9－3　DLL_ADD. dll 文件导出函数信息

通过 Depends 工具我们可以得知,DLL_ADD. dll 文件有两个导出函数,函数名分别为:

- Add@@YAHHH@Z;
- Sub。

这时候,读者可能会非常疑惑,明明声明的是导出 Add 函数,而实际导出的却是一个非常古怪的函数名"? Add@@YAHHH@Z"? 要回答这个问题,就需要弄明白 C 语言和 C++语言编写的动态链接库的不同。

A.3 C/C++语言动态链接库的不同

C++语言有一个很重要的特性即函数重载,相同名称参数不同的函数可以共存。如果按照 C 语言的标准,这时候就会出现编译错误。在 C++语言中为了避免这种编译错误的产生,编译器在编译时将函数的参数类型信息以及返回值类型信息加入到函数名中,这样代码中即使存在名字相同的重载函数,在经过编译后也能区分开;当然,调用时函数名也要经过相同的处理,以保证能找到对应的函数。读者可能会问,在 DLL_ADD. dll 文件中,为什么 Sub 函数的名称还是保持原样呢? 细心的读者也许已经发现了答案,即在 DLL_ADD. cpp 文件中,Sub 函数定义时的用 MYLIBAPI_C 宏修饰,而 Add 函数则用 MYLIBAPI_C_PLUS_PLUS 宏修饰。下面可以看一下这两个宏定义的不同之处:

```
#define MYLIBAPI_C_PLUS_PLUS __declspec(dllexport)
#define MYLIBAPI_C extern "C" __declspec(dllexport)
```

可以看出,与 MYLIBAPI_C_PLUS_PLUS 宏相比,MYLIBAPI_C 宏定义多了 extern "C"关键字修饰。extern "C" 关键字只用在 C++程序中,它用以通知编辑器其所修饰的变量和函数按照 C 语言方式编译和链接。到这里读者应当明白了为什么 DLL_ADD. dll 文件导出的 Sub 函数仍然保持函数名 Sub,正是 extern "C"关键字在起作用。

正是因为 C/C++动态链接库的不同,Matlab 编译器编译动态链接库的时候有 C 和 C++两种输出选项,在使用的时候也需要非常注意由于这种不同引起的问题。

A.4 动态链接库的调用方式

应用程序调用动态链接库时,有隐式和显示两种链接方式。

A.4.1 隐式链接

隐式链接是最常用的一种链接方式,使用起来比较简单。隐式链接需要用户提供动态链接库文件和动态链接库的导入库文件(*.lib 文件)。编译器通过导入库提供的信息,通知链接器所需的函数代码在 DLL 中。而链接器只需向可执行文件中添加信息,通知系统在进程启动时应在何处查找 DLL 代码即可。下面通过实例进一步说明隐式链接的实现方法。

① 首先建立一个 Visual C++ 6.0 Win32 Console Application 空工程 dll_call_hide。

② 将 DLL_ADD. dll 和 DLL_ADD. lib(这两个文件在 DLL_ADD 工程 Debug 目录下)复制到 dll_call_hide 工程目录下。

③ 然后通过选择 Project | Settings 菜单项打开的对话框中的 Link | Input 选项，将 DLL_ADD.lib 设置为链接库（如图 9－4 所示）。

④ 将 DLL_ADD.h 文件（DLL_ADD 工程目录下）复制到 dll_call_hide 工程目录下，并在工程主程序中增加对 DLL_ADD.h 头文件的引用。

⑤ 添加对 DLL_ADD.dll 调用的程序代码，代码如下所述。

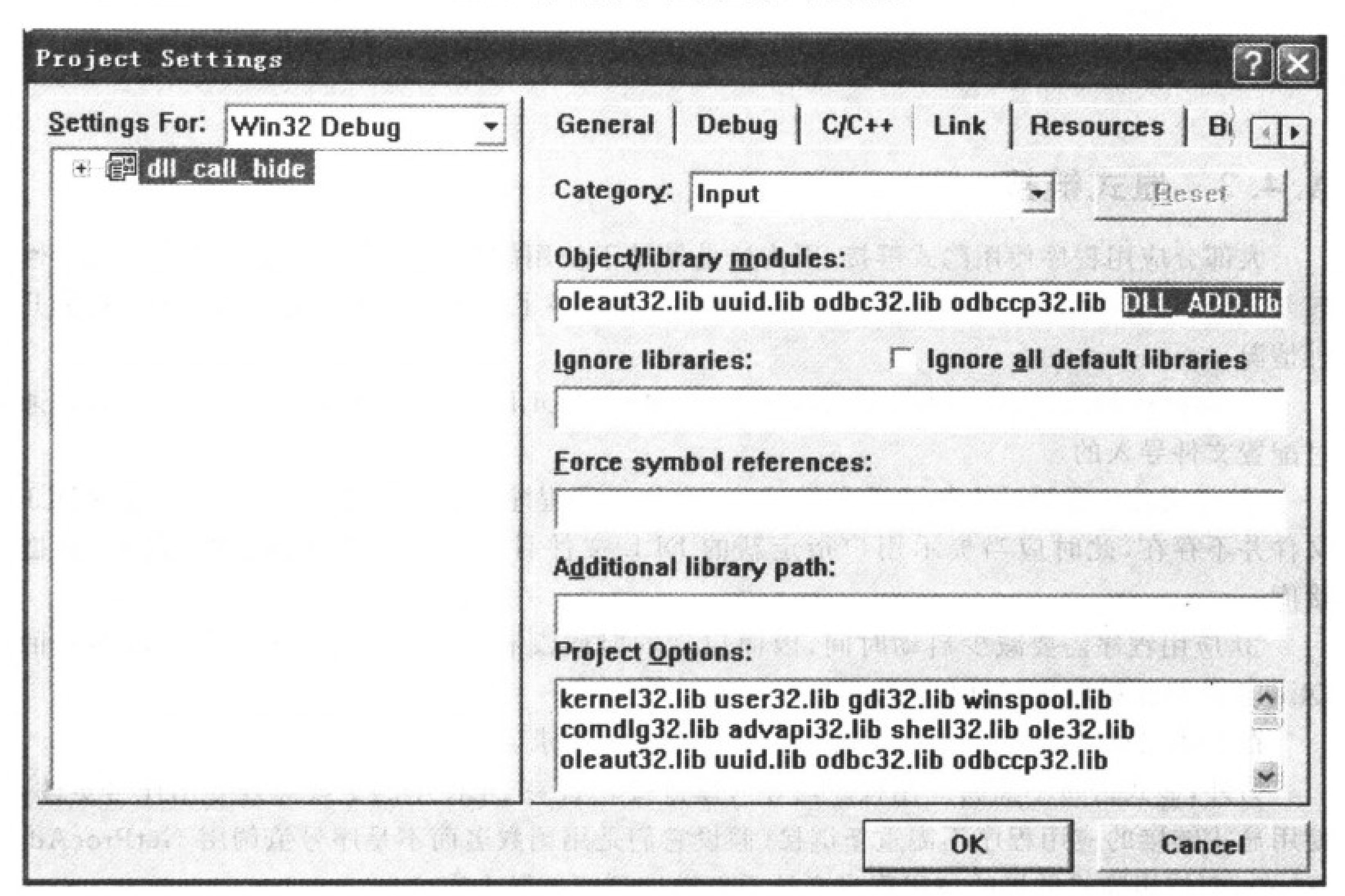


图 9－4　设置 DLL_ADD.lib 为链接库

```
/////////////////////////////////////////////////////////////////////////////////////
dll_call_hide.cpp
////////////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "stdio.h"
#include "DLL_ADD.h"


int main(int argc, char * argv[])
{
    int c;
    int a,b;
    a=1;
    b=2;
```

```
    c=Add(a,b);//调用动态链接库 DLL_ADD.dll 中的 add 函数
    printf("a=%d,b=%d,(a+b)=%d\n",a,b,c);
    c=Sub(a,b);//调用动态链接库 DLL_ADD.dll 中的 Sub 函数
    printf("a=%d,b=%d,(a-b)=%d\n",a,b,c);
    printf("Press Any Key To Quit! \n");
    getchar();
    return 0;
}
```

A.4.2 显式链接

大部分应用程序使用隐式链接,因为这是最易于使用的链接方法。但隐式链接的灵活性较差,因而在有些情况下,必须使用显示链接。下面列举了程序设计中需要使用显示链接的几种情况。

① 在被执行之前,应用程序并不知道需要加载的 DLL 文件名,比如 DLL 的文件名是通过配置文件导入的。

② 应用程序需要动态加载动态链接库,比如应用程序执行过程中发现自己所需的 DLL 文件并不存在,此时应当提示用户指定新的 DLL 文件目录并动态加载自己所需的动态链接库。

③ 应用程序需要减少启动时间,因而启动的时候没有必要加载所有的 DLL,因为有的 DLL 文件只在程序运行过程中使用。

在上述三种情况中,只有使用显示链接才能满足要求。

另外,显式链接不需将应用程序与导入库链接。如果 DLL 中的更改导致导出序号更改,使用显式链接的应用程序不需重新链接(假设它们是用函数名而不是序号值调用 GetProcAddress),而使用隐式链接的应用程序必须重新链接到新的导入库。

使用动态链接库显式链接的缺点是操作相对复杂。下面通过实例进一步说明显示式链接的实现方法。

① 首先建立一个 Visual C++ 6.0 Win32 Console Application 空工程 dll_call_apparent。

② 将 DLL_ADD.dll (在 DLL_ADD 工程 Debug 目录下)复制到 dll_call_ apparent 工程目录下。

③ 添加对 DLL_ADD.dll 显式调用的程序代码,代码如下所示。

```
//////////////////////////////////////////////////////////////////////////////////////////
dll_call_apparent.cpp
//////////////////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "stdio.h"
#include <windows.h>

//定义函数类型:输入为两个整型、输出为整型
typedef int (*MYFUN)(int,int);
```

```
int main(int argc,char * argv[])
{
    int c;
    int a,b;
    HINSTANCE hDLL = NULL;//DLL 句柄
    MYFUN Add = NULL,Sub = NULL;

    //加载 DLL 文件
    hDLL = LoadLibrary("DLL_ADD.dll");

    //获取 DLL 文件的 Add 函数
    Add = (MYFUN)GetProcAddress(hDLL,"? Add@@YAHHH@Z");
    //获取 DLL 文件的 Sub 函数
    Sub = (MYFUN)GetProcAddress(hDLL,"Sub");

    a = 1;
    b = 2;
    if(Add! = NULL)
    {
        c = Add(a,b);//调用动态链接库 DLL_ADD.dll add 函数
        printf("a = %d,b = %d,(a + b) = %d\n",a,b,c);
    }
    if(Sub! = NULL)
    {
        c = Sub(a,b);//调用动态链接库 DLL_ADD.dll 中的 Sub 函数
        printf("a = %d,b = %d,(a - b) = %d\n",a,b,c);
    }

    //释放
    FreeLibrary(hDLL);
    printf("Press Any Key To Quit! \n");
    getchar();
    return 0;
}
```

特别注意如下所述的源代码中比较重要的 Windows API 函数。

(1) LoadLibrary

LoadLibray 函数的参数是一个字符串指针，调用时需要填入需要加载的动态链接库的文件名(包含文件的目录)。

(2) GetProcAddress

GetProcAddress 函数从指定的动态库中查找指定名称的函数，如果查找成功则返回该函数的入口地址，如果失败则返回 NULL。可能读者已经注意到，GetProcAddress 函数中指定的函数名并不是 Add，而是？Add@@YAHHH@Z，这就和前面 A. 3 中讲到的 C/C++动态

链接库的不同联系起来了。应用 GetProcAddress 函数时，输入的函数名必须是编译后经过重命名的函数名，而不是源文件中定义的函数名。

(3) FreeLibrary

调用 FreeLibrary 函数释放指定的动态链接库(由动态链接库的句柄标志)，读者需要注意的是 LoadLibrary 和 FreeLibrary 一定要配对使用，否则会造成动态链接库没有及时释放而导致内存泄露。

除了 Windows API 函数以外，在 dll_call_apparent. cpp 源代码中，还定义了函数类型 MYFUN 和句柄 hDLL。其中定义 MYFUN 函数类型是为引用 DLL_ADD. dll 的两个函数 Add 和 Sub 更加方便；定义 hDll 句柄用以存储 DLL_ADD. dll 加载时返回的实例句柄(句柄—Handle，可以联想为球拍的手柄，抓住了手柄就能掌握球拍，当然，知道了 DLL 文件的句柄，理论上就可以对其进行任何合理的操作，实际上句柄是一个内存地址，在对应的内存中存放的是 DLL 文件的信息)。



参考文献

1 Kruglinski D J. Visual C++技术内幕[M]. 潘爱民,王国印译. 第 4 版. 北京:清华大学出版社,2001

2 潘爱民. COM 原理及应用[M]. 北京:清华大学出版社,2001

3 侯俊杰. 深入浅出 MFC[M]. 武汉:华中科技大学出版社,2001

4 苏金明. MATLAB 与外部程序接口[M]. 北京:电子工业出版社,2004

5 余啸海. Matlab 接口技术与应用[M]. 北京:国防工业出版社,2004

6 林岚. Matcom:Matlab 与 C++结合的有效途径[J]. 微处理机,1999(2)

7 袁勇,吴禄慎,潘俊伟. 实现 Matlab 与 VC++混合编程的有效途径[J]. 飞机设计,2003(1)

8 何晓涛,于春田. VC 调用 MATLAB 的方法[J]. 河北科技大学学报,2003,24(1)

9 况志军,陈进,徐征. 在 VC++中通过调用 MATLAB 实现回归分析[J]. 华东交通大学学报,2004,21(1)

